



GE VERNOVA

PROFICY® SOFTWARE & SERVICES

CIMPLICITY

Basic Control Engine
and Scripting Reference

Proprietary Notice

The information contained in this publication is believed to be accurate and reliable. However, GE Vernova assumes no responsibilities for any errors, omissions or inaccuracies. Information contained in the publication is subject to change without notice.

No part of this publication may be reproduced in any form, or stored in a database or retrieval system, or transmitted or distributed in any form by any means, electronic, mechanical photocopying, recording or otherwise, without the prior written permission of GE Vernova. Information contained herein is subject to change without notice.

© 2024 GE Vernova and/or its affiliates. All rights reserved.

Trademark Notices

“VERNOVA” is a registered trademark of GE Vernova. “GE VERNOVA” is a registered trademark of GE Aerospace exclusively licensed to GE Vernova. The terms “GE” and the GE Monogram are trademarks of GE Aerospace and are used with permission.

Microsoft® is a registered trademark of Microsoft Corporation, in the United States and/or other countries.

All other trademarks are the property of their respective owners.

We want to hear from you. If you have any comments, questions, or suggestions about our documentation, send them to the following email address:
doc@ge.com

Contents

- Chapter 1. Event Editor.....32**
 - About the Event Editor..... 32
 - Event Editor Configuration..... 33
 - Event Editor Configuration.....33
 - Step 1. Open the Event Editor.....34
 - Step 2. Review Event Editor Features..... 35
 - Step 3. Configure an Event..... 41
 - Step 4. Create an Action..... 60
 - Step 5. Associate Actions with an Event..... 74
 - Step 6. Work with Existing Events and Actions.....75
 - Configure Multiple Event Manager Instances in a Project..... 97
 - About Multiple Event Manager Resident Process (EMRP)..... 97
 - Add a New EMRP Instance.....98
 - Associate Event Manager Instance to an Event.....103
 - Check if the New Event Manager Instance is Running..... 104
 - Test the Newly Configured Event in Basic Control Engine User Interface (BCEUI)..... 106
 - Optimize Event Editor Performance.....108
- Chapter 2. Script Editors..... 110**
 - About Script Editors..... 110
 - About the Program Editor.....110
 - Open the Program Editor..... 111
 - Open the Program Editor..... 111
 - Option 1. Open a Blank Program Editor.....111
 - Option 2. Open the Program Editor with an Existing Script.....113
 - Program Editor Window Components..... 114
 - Program Editor Window Components.....114
 - Program Editor Menu Functions.....116

Program Editor Toolbars and Status Bar.....	121
Program Editor Shortcut Keys.....	124
Set String and Stack Space.....	125
Program Editor: Edit Programs.....	127
Program Editor: Edit Programs.....	127
1. Program Editor: Navigate within a Script.....	128
2. Program Editor: Text Procedures.....	129
3. Program Editor: Point Tools.....	137
4. Program Editor: Alarm Tools.....	143
5. Program Editor: Log Status Tool.....	145
6. Program Editor: Add Comments to a Script.....	146
7. Program Editor: Enter a Statement across Multiple Lines.....	147
8. Program Editor: Check the Syntax of a Script.....	147
9. Program Editor: Add Dialog Boxes to a Script.....	148
Dialog Editor.....	148
Dialog Editor.....	148
1. Use the Dialog Editor.....	149
2. Create a Custom Dialog Box.....	156
3. Edit a Custom Dialog Box.....	164
4. Insert/Paste a Dialog Box Template Code into a Script.....	191
5. Edit an Existing Dialog Box.....	195
6. Test a Dialog Box.....	202
7. Exit from the Dialog Editor.....	206
8. Use a Custom Dialog Box in a Script.....	208
9. Use a Dynamic Dialog Box in a Script.....	213
Debug Scripts.....	217
Debug Scripts.....	217
1. Fabricate Event Information.....	218
2. Step through Scripts.....	220

3. Use Breakpoints.....	225
4. Perform Traces in Scripts.....	229
5. Use a Watch Variable.....	232
Run a Program.....	240
Error Messages.....	241
Error Messages.....	241
1. Visual Basic Compatible Error Messages.....	242
2. Basic Control Engine-Specific Error Messages.....	245
3. Error Message List.....	246
Chapter 3. CimScriptIDE Editor.....	252
About the CimScriptIDE Editor.....	252
1. Open the CimScriptIDE Editor.....	253
1. Open the CimScriptIDE Editor.....	253
1.1. Create a New C# or VB .NET Script.....	253
1.2. Open an Existing C# or VB .NET Script.....	255
2. CimScriptIDE Editor: Overview.....	256
2. CimScriptIDE Editor: Overview.....	256
2.1. CimScriptIDE Editor: Menus.....	258
2.2. CimScriptIDE Editor: Toolbars and Status Bar.....	262
2.3. CimScriptIDE Editor: Class View Pane.....	263
2.4. CimScriptIDE Editor: Right-Pane.....	264
3. Technical Reference: CimScriptIDE Editor.....	265
3. Technical Reference: CimScriptIDE Editor.....	265
3.1. CimScriptIDE Debugging in Visual Studio.....	266
3.2. Attach Additional .NET Assembly References.....	267
Chapter 4. Basic Control Engine Language Reference.....	270
Using the Basic Control Engine Language Reference.....	270
Scripting Language Reference.....	271
About the Basic Control Syntax.....	272

Language Elements by Category.....	273
Language Elements By Category.....	273
Arrays.....	274
Clipboard.....	275
Comments.....	275
Comparison Operators.....	275
Controlling other Programs.....	275
Controlling Program Flow.....	276
Controlling the Operating Environment.....	277
Conversion.....	277
Data Types.....	278
Database.....	279
Date/time.....	279
DDE.....	280
Error Handling.....	280
File I/O.....	281
File System.....	282
Financial.....	282
Getting information from Basic Control Engine.....	283
INI Files.....	283
Logical/binary Operators.....	284
Math.....	284
Miscellaneous.....	285
Numeric Operators.....	285
Objects.....	285
Parsing.....	286
Predefined Dialogs.....	286
Printing.....	286
Procedures.....	287

String Operators.....	287
Strings.....	287
User Dialogs.....	288
Variables and Constants.....	289
Variants.....	290
Symbols.....	290
Symbols.....	290
' (keyword).....	291
- (operator).....	291
#Const (directive).....	293
#if...Then...#Else (directive).....	294
& (operator).....	297
() (keyword).....	297
* (operator).....	298
. (keyword).....	299
/ (operator).....	300
\ (operator).....	301
^ (operator).....	302
_ (keyword).....	303
+ (operator).....	303
< (operator).....	305
<= (operator).....	305
<> (operator).....	305
= (operator).....	305
= (statement).....	306
> (operator).....	306
>= (operator).....	306
A.....	307
A.....	307

Abs (function).....	308
And (operator).....	309
AnswerBox (function).....	310
Any (data type).....	312
AppActivate (statement).....	313
AppClose (statement).....	314
AppFind, AppFind\$ (functions).....	315
AppGetActive\$ (function).....	316
AppGetPosition (statement).....	317
AppGetState (function).....	318
AppHide (statement).....	319
AppList (statement).....	320
AppMaximize (statement).....	320
AppMinimize (statement).....	321
AppMove (statement).....	322
AppRestore (statement).....	323
AppSetState (statement).....	324
AppShow (statement).....	325
AppSize (statement).....	326
AppType (function).....	327
ArrayDims (function).....	328
Arrays (topic).....	329
ArraySort (statement).....	331
Asc, AscB, AscW (functions).....	332
AskBox, AskBox\$ (functions).....	333
AskPassword, AskPassword\$ (functions).....	335
Atn (function).....	337
B.....	337
B.....	337

Basic.Architecture\$ (property).....	338
Basic.Capability (method).....	339
Basic.CodePage (property).....	340
Basic.Eoln\$ (property).....	340
Basic.FreeMemory (property).....	341
Basic.HomeDir\$ (property).....	341
Basic.Locale\$ (property).....	342
Basic.OperatingSystem\$ (property).....	343
Basic.OperatingSystemVendor\$ (property).....	343
Basic.OperatingSystemVersion\$ (property).....	344
Basic.OS (property).....	345
Basic.PathSeparator\$ (property).....	346
Basic.Processor\$ (property).....	346
Basic.ProcessorCount\$ (property).....	347
Basic.Version\$ (Property).....	347
Beep (statement).....	348
Begin Dialog (statement).....	348
Boolean (data type).....	351
ByRef (keyword).....	352
ByVal (keyword).....	352
C.....	353
C.....	353
Call (statement).....	355
CDBl (function).....	356
CBool (function).....	356
CCur (function).....	357
CDate, CVDate (functions).....	358
ChDir (statement).....	359
ChDrive (statement).....	359

CheckBox (statement).....	360
Choose (function).....	362
Chr, Chr\$, ChrB, ChrB\$, ChrW, ChrW\$ (functions).....	362
CInt (function).....	364
CancelButton (statement).....	365
Clipboard\$ (function).....	367
Clipboard \$ (statement).....	367
Clipboard.Clear (method).....	368
CreateObject (function).....	368
Clipboard.GetFormat (method).....	370
Clipboard .GetText (method).....	371
Clipboard .SetText (method).....	371
CLng (function).....	372
Close (statement).....	373
ComboBox (statement).....	373
Command, Command\$ (functions).....	375
Comparison Operators (topic).....	376
Const (statement).....	378
Constants (topic).....	380
Cos (function).....	385
CSng (function).....	385
CStr (function).....	386
CurDir, CurDir\$ (functions).....	387
Currency (data type).....	387
CVar (function).....	388
CVErr (function).....	389
Comments (topic).....	390
D.....	390
D.....	390

Date (data type).....	392
Date, Date\$ (functions).....	393
Date, Date\$ (statements).....	394
DateAdd (function).....	395
DateDiff (function).....	397
DatePart (function).....	398
DateSerial (function).....	400
DateValue (function).....	401
Day (function).....	401
DDB (function).....	402
DDEExecute (statement).....	403
DDEInitiate (function).....	404
DDEPoke (statement).....	405
DDERequest, DDERequest\$ (functions).....	407
DDESend (statement).....	408
DDETerminate (statement).....	409
DDETerminateAll (statement).....	410
DDETimeout (statement).....	411
Declare (statement).....	412
DefType (statement).....	421
DeleteSetting (statement).....	423
Dialog (function).....	424
Dialog (statement).....	426
Dim (statement).....	426
Dir, Dir\$ (functions).....	427
DiskDrives (statement).....	428
DiskFree (function).....	429
DlgCaption (function).....	430
DlgCaption (statement).....	430

DlgControlId (function).....	431
DlgEnable (function).....	432
DlgEnable (statement).....	434
DlgFocus (function).....	435
DlgFocus (statement).....	436
DlgListBoxArray (function).....	437
DlgListBoxArray (statement).....	439
DlgProc (function).....	440
DlgSetPicture (statement).....	443
DlgText (statement).....	444
DlgText\$ (function).....	446
DlgValue (function).....	448
DlgValue (statement).....	449
DlgVisible (function).....	451
DlgVisible (statement).....	451
Do...Loop (statement).....	454
DoEvents (function).....	456
DoEvents (statement).....	457
Double (data type).....	458
DropListBox (statement).....	458
E.....	460
E.....	460
ebAbort (constant).....	463
ebAbortRetryIgnore (constant).....	464
ebApplicationModal (constant).....	464
ebArchive (constant).....	465
ebBold (constant).....	465
ebBoldItalic (constant).....	466
ebBoolean (constant).....	466

ebCancel (constant).....	467
ebCritical (constant).....	467
ebCurrency (constant).....	468
ebDataObject (constant).....	468
ebDate (constant).....	469
ebDefaultButton1 (constant).....	469
ebDefaultButton2 (constant).....	469
ebDefaultButton3 (constant).....	470
ebDirectory (constant).....	470
ebDos (constant).....	471
ebDouble (constant).....	471
ebEmpty (constant).....	471
ebError (constant).....	472
ebExclamation (constant).....	473
ebHidden (constant).....	473
ebIgnore (constant).....	474
ebInformation (constant).....	474
ebInteger (constant).....	474
ebItalic (constant).....	475
ebLong (constant).....	476
ebNo (constant).....	476
ebNone (constant).....	476
ebNormal (constant).....	477
ebNull (constant).....	478
ebObject (constant).....	478
ebOK (constant).....	479
ebOKCancel (constant).....	479
ebOKOnly (constant).....	479
ebQuestion (constant).....	480

ebReadOnly (constant).....	480
ebRegular (constant).....	481
ebRetry (constant).....	481
ebRetryCancel (constant).....	482
ebSingle (constant).....	482
ebString (constant).....	483
ebSystem (constant).....	483
ebSystemModal (constant).....	484
ebVariant (constant).....	484
ebVolume (constant).....	484
ebYes (constant).....	485
ebYesNo (constant).....	485
ebYesNoCancel (constant).....	486
Empty (constant).....	486
End (statement).....	487
End Dialog (statement).....	487
Environ, Environ\$ (functions).....	488
EOF (function).....	488
Eqv (operator).....	489
Erase (statement).....	490
Erl (function).....	492
Err (function).....	492
Err (statement).....	493
Error, Error\$ (functions).....	493
Error (statement).....	494
Error Handling (topic).....	495
Err.Clear (method).....	495
Err.Description (property).....	497
Err.HelpContext (property).....	498

Err.HelpFile (property).....	499
Err.LastDLLError (property).....	500
Err.Number (property).....	501
Err.Raise (method).....	503
Err.Source (property).....	504
Exit Do (statement).....	505
Exit For (statement).....	506
Exit Function (statement).....	507
Exit Sub (statement).....	507
Exp (function).....	508
Expression Evaluation (topic).....	509
F.....	510
F.....	510
False (constant).....	511
FileAttr (function).....	512
FileCopy (statement).....	513
FileDateTime (function).....	514
FileDirs (statement).....	515
FileExists (function).....	516
FileLen (function).....	517
FileList (statement).....	517
FileParse\$ (function).....	520
Fix (function).....	521
For Each...Next (statement).....	522
For...Next (statement).....	524
Format, Format\$ (functions).....	525
FreeFile (function).....	532
Function...End Function (statement).....	533
Fv (function).....	533

G.....	534
G.....	534
Get (statement).....	535
GetAllSettings (function).....	537
GetAttr (function).....	538
GetObject (function).....	540
GetSetting (function).....	541
Global (statement).....	543
GoSub (statement).....	543
Goto (statement).....	544
GroupBox (statement).....	544
H.....	546
H.....	546
HelpButton (statement).....	546
Hex, Hex\$ (functions).....	547
HLine (statement).....	548
Hour (function).....	549
HPage (statement).....	549
HScroll (statement).....	550
HWND (object).....	550
HWND.Value (property).....	551
I.....	552
I.....	552
If...Then...Else (statement).....	553
IIf (function).....	555
IMEStatus (function).....	556
Imp (operator).....	557
Input# (statement).....	559
Input, Input\$, InputB, InputB\$ (functions).....	560

InputBox, InputBox\$ (functions).....	562
InStr, InStrB (functions).....	563
Int (function).....	565
Integer (data type).....	566
IPmt (function).....	566
IRR (function).....	568
Is (operator).....	570
IsDate (function).....	571
IsEmpty (function).....	572
IsError (function).....	572
IsMissing (function).....	573
IsNull (function).....	574
IsNumeric (function).....	574
IsObject (function).....	575
IsWebSpaceSession (function).....	576
Item\$ (function).....	576
ItemCount (function).....	577
K.....	578
K.....	578
Keywords (topic).....	578
Kill (statement).....	579
L.....	580
L.....	580
LBound (function).....	581
LCase, LCase\$ (functions).....	582
Left, Left\$, LeftB, LeftB\$ (functions).....	582
Len (function).....	584
Let (statement).....	586
Like (operator).....	587

Line Input# (statement).....	588
Line\$ (function).....	589
LineCount (function).....	590
Line Numbers (topic).....	591
ListBox (statement).....	591
Literals (topic).....	593
Loc (function).....	594
Lock (statement).....	595
Lof (function).....	597
Log (function).....	598
Long (data type).....	598
LSet (statement).....	599
LTrim, LTrim\$ (functions).....	600
M.....	600
M.....	600
Main (statement).....	601
MCI (function).....	601
Mid, Mid\$, MidB, MidB\$ (functions).....	603
Mid, Mid\$, MidB, MidB\$ (statements).....	605
Minute (function).....	606
MIRR (function).....	606
MkDir (statement).....	608
Mod (operator).....	609
Month (function).....	610
Msg.Close (method).....	610
Msg.Open (method).....	611
Msg.Text (property).....	612
Msg.Thermometer (property).....	613
MsgBox (function).....	614

MsgBox (statement).....	617
N.....	618
N.....	618
Name (statement).....	619
Named Parameters (topic).....	620
Net.AddCon (method).....	621
Net.Browse\$ (method).....	622
Net.CancelCon (method).....	623
Net.GetCaps (method).....	624
Net.GetCon\$ (method).....	625
Net.User\$ (property).....	626
New (keyword).....	627
Not (operator).....	627
Nothing (constant).....	628
Now (function).....	629
NPer (function).....	629
Npv (function).....	630
Null (constant).....	632
O.....	632
O.....	632
Object (data type).....	633
Objects (topic).....	634
Oct, Oct\$ (functions).....	637
OKButton (statement).....	638
On Error (statement).....	640
Open (statement).....	642
Option Default (statement).....	645
Option Explicit (statement).....	645
OpenFilename\$ (function).....	646

Operator Precedence (topic).....	648
Operator Precision (topic).....	649
Option Base (statement).....	649
Option Compare (statement).....	649
Option CStrings (statement).....	651
OptionButton (statement).....	652
OptionGroup (statement).....	653
Or (operator).....	654
P.....	656
P.....	656
Pi (constant).....	657
Picture (statement).....	657
PictureBox (statement).....	659
Pmt (function).....	661
PopupMenu (function).....	662
PPmt (function).....	663
Print (statement).....	664
Print# (statement).....	666
Private (statement).....	668
Public (statement).....	670
PushButton (statement).....	671
Put (statement).....	673
Pv (function).....	675
Q.....	676
QueEmpty (statement).....	676
R.....	677
R.....	677
Random (function).....	678
Randomize (statement).....	679

Rate (function).....	680
RCPDownload (statement).....	681
RCPDownloadEx (function).....	682
RCPGroupExport (statement).....	683
RCPGroupExportEx (function).....	683
RCPGroupImport (statement).....	684
RCPGroupImportEx (function).....	684
RCPUpload (statement).....	685
RCPUploadEx (function).....	686
ReadIni\$ (function).....	687
ReadIniSection (statement).....	687
Redim (statement).....	688
Rem (statement).....	690
Reset (statement).....	690
Resume (statement).....	691
Return (statement).....	692
Right, Right\$, RightB, RightB\$ (functions).....	692
Rmdir (statement).....	694
Rnd (function).....	694
RSet (statement).....	695
RTrim, RTrim\$ (functions).....	696
S.....	697
S.....	697
SaveFilename\$ (function).....	699
SaveSetting (statement).....	701
Screen.DlgBaseUnitsX (property).....	702
Screen.DlgBaseUnitsY (property).....	702
Screen.Height (property).....	703
Screen.TwipsPerPixelX (property).....	703

Screen.TwipsPerPixelY (property).....	704
Screen.Width (property).....	704
Second (function).....	705
Seek (function).....	706
Seek (statement).....	707
Select...Case (statement).....	708
SelectBox (function).....	709
SendKeys (statement).....	711
Set (statement).....	714
SetAttr (statement).....	715
Sgn (function).....	716
Shell (function).....	717
Sin (function).....	718
Single (data type).....	718
Sleep (statement).....	719
Sln (function).....	720
Space, Space\$ (functions).....	720
Spc (function).....	721
SQLBind (function).....	722
SQLClose (function).....	723
SQLError (function).....	724
SQLExecQuery (function).....	726
SQLGetSchema (function).....	727
SQLOpen (function).....	730
SQLQueryTimeout (statement).....	731
SQLRequest (function).....	731
SQLRetrieve (function).....	733
SQLRetrieveToFile (function).....	735
Sqr (function).....	736

Stop (statement).....	737
Str, Str\$ (functions).....	737
StrComp (function).....	738
StrConv (function).....	740
String (data type).....	742
String, String\$ (functions).....	743
Sub...End Sub (statement).....	744
Switch (function).....	744
SYD (function).....	745
System.Exit (method).....	746
System.FreeMemory (property).....	747
System.FreeResources (property).....	747
System.MouseTrails (method).....	748
System.Restart (method).....	748
System.TotalMemory (property).....	748
System.WindowsDirectory\$ (property).....	749
System.WindowsVersion\$ (property).....	749
T.....	750
T.....	750
Tab (function).....	750
Tan (function).....	751
Text (statement).....	752
TextBox (statement).....	753
Time, Time\$ (functions).....	755
Time, Time\$ (statements).....	756
Timer (function).....	757
TimeSerial (function).....	757
TimeValue (function).....	758
Trim, Trim\$, LTrim, LTrim\$, RTrim, RTrim\$ (functions).....	758

True (constant).....	760
Type (statement).....	761
TypeOf (function).....	762
TypeName (function).....	763
U.....	764
U.....	764
UBound (function).....	765
UCase, UCase\$ (functions).....	766
Unlock (statement).....	766
User-Defined Types (topic).....	768
V.....	769
V.....	769
Val (function).....	770
Variant (data type).....	771
VarType (function).....	773
Viewport.Clear (method).....	774
Viewport.Close (method).....	775
Viewport.Open (method).....	775
VLine (statement).....	776
VPage (statement).....	777
VScroll (statement).....	778
W.....	778
W.....	778
Weekday (function).....	779
While...Wend (statement).....	780
Width# (statement).....	781
WinActivate (statement).....	782
WinClose (statement).....	783
WinFind (function).....	785

WinList (statement).....	785
WinMaximize (statement).....	786
WinMinimize (statement).....	787
WinMove (statement).....	788
WinRestore (statement).....	789
WinSize (statement).....	790
Word\$ (function).....	792
WordCount (function).....	793
Write# (statement).....	793
Writeln (statement).....	795
X.....	795
X or (operator).....	795
Y.....	797
Year (function).....	797
CIMPLICITY Extensions to Basic.....	797
CIMPLICITY Extensions to Basic.....	797
Acquire (function).....	804
Acquire, Release (statements).....	805
AlarmGenerate (statement).....	806
AlarmGenerateEx (statement).....	808
AlarmUpdate (statement).....	813
AlarmUpdateCA (statement).....	814
AlarmUpdateEx (statement).....	816
ChangePassword (statement).....	820
CimChangeApprovalData (Object).....	821
CimEMAlarmEvent.AlarmID (property, read).....	821
CimEMAlarmEvent.FinalState (property, read).....	822
CimEMAlarmEvent.GenTime (property, read).....	822
CimEMAlarmEvent.Message (property, read).....	823

CimEMAlarmEvent (object).....	823
CimEMAlarmEvent.PrevState (property, read).....	823
CimEMAlarmEvent.RefID (property, read).....	824
CimEMAlarmEvent.RegAction (property, read).....	824
CimEMAlarmEvent.ResourceID (property, read).....	825
CimEMEvent.ActionID (property, read).....	825
CimEMEvent.AlarmEvent (function).....	825
CimEMEvent.EventID (property, read).....	826
CimEMEvent (object).....	826
CimEMEvent.ObjectID (property, read).....	826
CimEMEvent.PointEvent.....	827
CimEMEvent.TimeStamp (property, read).....	827
CimEMEvent.Type (property, read).....	827
CimEMPointEvent.Id.....	828
CimEMPointEvent (object).....	829
CimEmPointEvent.Quality (property, read).....	829
CimEmPointEvent.QualityAlarmed (property, read).....	830
CimEmPointEvent.QualityAlarms_Enabled (property, read).....	830
CimEmPointEvent.QualityDisable_Write (property, read).....	830
CimEmPointEvent.QualityLast_Upd_Man (property, read).....	831
CimEmPointEvent.QualityManual_Mode (property, read).....	831
CimEmPointEvent.QualityIs_In_Range (property, read).....	831
CimEmPointEvent.QualityStale_Data (property, read).....	832
CimEmPointEvent.QualityIs_Available (property, read).....	832
CimEMPointEvent.State (property, read).....	833
CimEMPointEvent.TimeStamp (property, read).....	833
CimEmPointEvent.UserFlags (property, read).....	833
CimEMPointEvent.Value (property, read).....	834
CimGetEMEvent (function).....	834

CimIsMaster (function).....	835
CimLogin (statement).....	835
CimLogout (statement).....	835
CimProjectData.Attributes (property, read/write).....	836
CimProjectData.Filters (property, read/write).....	836
CimProjectData.GetNext (function).....	837
CimProjectData.Entity (property, read/write).....	838
CimProjectData (object).....	852
CimProjectData.Project (property, read/write).....	853
CimProjectData.Reset (method).....	854
CimRemoveUnusedPoints (method).....	854
DoQINTMath (function).....	854
DoUQINTMath (function).....	855
GetCurTimeHR (function).....	856
GetKey (function).....	857
GetMemoryInfoSymbolSpace (statement).....	857
GetMemoryInfoStringSpaceHandles (statement).....	859
GetMemoryInfoStringSpace (statement).....	861
GetMemoryInfoPublicSpace (statement).....	862
GetSystemWindowsDirectory (function).....	864
GetTimeComponentsHR (function).....	864
GetTSSessionId (function).....	865
IsTerminalServices (function).....	866
LogStatus (property, read/write).....	866
Point.AlarmAck (property, read).....	867
Point.Cancel (method).....	867
Point.ChangeApproval (property, write).....	868
Point.ChangeApprovalInfo (property, read).....	869
Point.DataType (property, read).....	870

Point.DisplayFormat (property, read).....	871
Point.DownloadPassword (property, read).....	871
Point.Elements (property, read).....	871
Point.EnableAlarm (method).....	872
Point.Enabled (property, read).....	872
Point.EuLabel (property, read).....	873
Point.Get (statement).....	873
Point.GetArray (statement).....	873
Point.GetNext (function).....	875
Point.GetNext (statement).....	875
Point.GetQuadIntValue (function).....	876
Point.GetRawArray (statement).....	877
Point.GetTimeStampHR (statement).....	878
Point.GetValue (property, read).....	879
Point.HasEuConv (property, read).....	879
Point.Id (property, read/write).....	880
Point.InUserView (property, read).....	880
Point.Length (property, read).....	881
Point (object).....	881
Point.OnAlarm (statement).....	882
Point.OnAlarmAck (statement).....	884
Point.OnChange (statement).....	884
Point.OnTimed (statement).....	885
Point.PointTypeId (property, read).....	886
Point.QuadValueAsString (property, read).....	886
Point.QuadValueAsString (property, write).....	887
Point.Quality (property, read).....	887
Point.QualityAlarmed (property, read).....	887
Point.QualityAlarms_Enabled (property, read).....	888

Point.QualityDisable_Write (property, read).....	888
Point.QualityIs_Available (property, read).....	889
Point.QualityIs_In_Range (property, read).....	889
Point.QualityLast_Upd_Man (property, read).....	889
Point.QualityManual_Mode (property, read).....	890
Point.QualityStale_Data (property, read).....	890
Point.RawValue (property, read/write).....	891
Point.ReadOnly (property, read).....	893
Point.Set (statement).....	893
Point.SetArray (statement).....	894
Point.SetElement (statement).....	895
Point.SetNoAudit (statement).....	896
Point.SetpointPriv (property, read).....	896
Point.SetQuadIntValue (function).....	897
Point.SetRawArray (statement).....	897
Point.SetValue (property, write).....	899
Point.State (property, read).....	899
Point (subject).....	900
Point.TimeStamp (property, read).....	904
Point.TimeStampHR (property, read).....	905
Point.UserFlags (property, read).....	905
Point.Value (property, read/write).....	906
PointGet (function).....	906
PointGetMultiple (function).....	908
PointGetNext (function).....	910
PointSet (statement).....	913
PointSetMultiple (function).....	914
PointSetMultipleEx (function).....	916
SetTimecomponentsHR (function).....	918

QINTFromString (function).....	920
StringFromQINT (function).....	920
StringFromUQINT (function).....	921
Trace (statement).....	922
TraceEnable/TraceDisable (statement).....	922
UQINTFromString (function).....	923
Chapter 5. Basic Control Engine User Interface.....	925
About the BCEUI.....	925
Open the BCEUI Window.....	925
BCEUI Menus.....	927
BCEUI Menus.....	927
BCEUI File Menu.....	927
BCEUI Events Menu.....	928
BCEUI Scripts Menu.....	928
BCEUI View Menu.....	928
BCEUI Help Menu.....	929
BCEUI Window Pop-up Menu.....	929
BCEUI Toolbar.....	929
BCEUI Shortcut Keys.....	930
BCEUI Viewer.....	930
BCEUI Viewer.....	930
1. Select Events in the Browser.....	931
2. Toggle the Auto Browse.....	932
3. Connect to a Project.....	933
4. Select Events.....	933
5. Use the Event List.....	933
6. Set the Maximum Number of Completed Actions.....	935
7. Add Events to the View.....	935
8. Remove Events from the View.....	935

9. Trigger Events.....	935
Control Scripts.....	936
Control Scripts.....	936
Pause Scripts.....	936
Resume Scripts.....	937
Stop Scripts.....	938
Chapter 6. Action Calendar.....	940
About the Action Calendar.....	940
Planning for the Action Calendar.....	941
What the Action Calendar Does.....	941
When to use Other CIMPLICITY Tools.....	942
Action Calendar Interface Overview.....	942
About the Action Calendar Scheduler.....	945
Action Calendar Planning Configuration.....	945
Production Shifts and Days.....	952
Configuration Changes Incorporated into the System.....	953
Sample Factory Configuration Example.....	953
Configuring the Action Calendar.....	959
Action Calendar at a Glance.....	959
Action Calendar Data Entry Overview.....	963
Factory Action Calendar Schedule Example.....	963
Area Configuration.....	964
Day Type Legend.....	967
Event Configuration.....	975
Schedules.....	982
Security.....	992
Procedure to set the Day Start Time.....	994
Command Line Parameters.....	994
Chapter 7. Python Scripting.....	996

Overview.....	996
Python APIs.....	997
Additional Packages.....	997
Install Third Party Packages.....	998
Python Scripts for Event Manager Actions.....	1001
Writing Standalone Python Scripts.....	1003
Writing Common Code for Python Scripts.....	1007
Testing Python Event Manager Scripts.....	1009

Chapter 1. Event Editor

About the Event Editor

You use the Event Editor to define actions to take in response to events that occur in a process. One event may invoke multiple actions, or one action may be invoked by many events.

An event can be defined as a changing point or alarm state, or even a time of day.

Based on an event, you can perform the following actions:

- Set point values
- Acknowledge or clear alarms
- Create log file entries
- Invoke specific user-defined actions
- Invoke Basic Control Engine scripts to execute user-defined logic

At run-time, the Basic Control Engine monitors for events and executes the configured actions.

The Basic Control Engine is based on a multi-threaded design, which allows the system to invoke and execute multiple Basic Control Engine scripts concurrently.

The order of execution of event actions is a sequential execution from top to bottom.

Note: The script is run in parallel with all actions that are being executed for the event. In other words, the Basic Control Engine does not wait for the script to complete before it initiates the next action defined for the event.

Any action can be invoked by any event.

A few of the ways actions and events may be combined are:

Combined Actions and Events	Description
Point Actions Based on Point Events	Passes information between points.
Point Actions Based on Alarm Events	Allow a physical indication of an alarm, such as activating a light on a control panel.
Events Whose Actions Call A User-Defined Routine or Script	Defines custom functions that are invoked in response to configured system events.

**Note:**

The Basic Control Engine calls a startup script when the Event Manager starts up and a termination script when it shuts down. These scripts are initially null (that is, they do not do anything). You can use these scripts to perform initialization and termination tasks, such as restoring and saving the value of a global variable. The two scripts are:

- EM_INIT.BCL
- EM_TERM.BCL

You will find copies of these scripts in your project's \scripts directory.

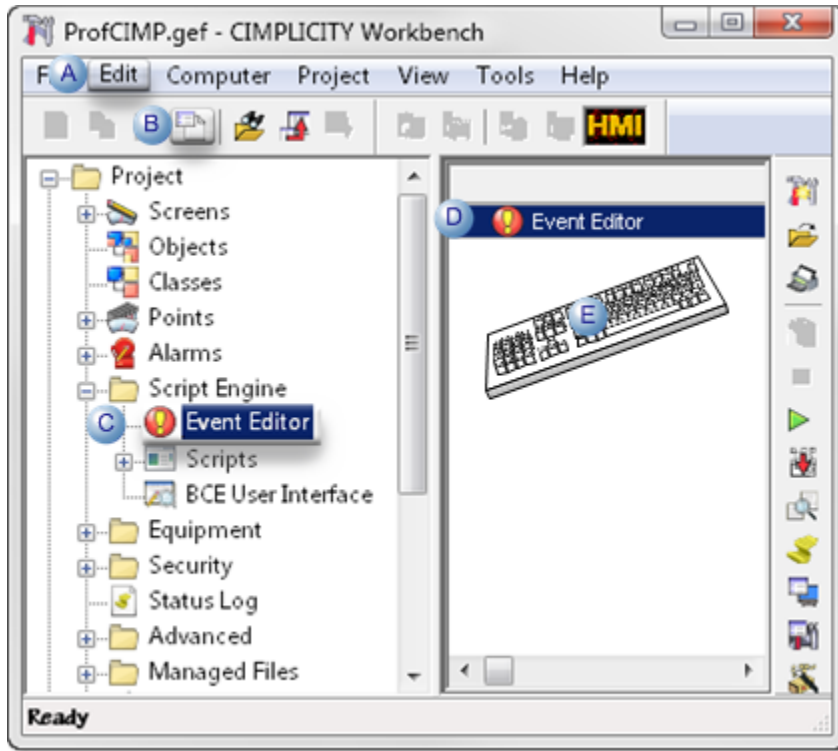
Event Editor Configuration

Event Editor Configuration

Step 1 <i>(on page 34)</i>	Open the Event Editor.
Step 2 <i>(on page 35)</i>	Review Event Editor features.
Step 3 <i>(on page 41)</i>	Create an event.
Step 4 <i>(on page 60)</i>	Create an action.
Step 5 <i>(on page 74)</i>	Associate actions with an event.
Step 6 <i>(on page 75)</i>	Copy, filter, select display fields for events and actions.

Step 1. Open the Event Editor

1. Select **Project>Basic Control Engine>Event Editor** in the Workbench left pane.
2. Select **Event Editor** in the Workbench right pane.
3. Do one of the following.



A Click Edit>Properties on the Workbench menu bar.	
B Click the Properties button on the Workbench toolbar.	
C In the Workbench left pane:	
Either	Or
Double click Event Editor .	<ol style="list-style-type: none"> a. Right-click Event Editor. b. Select Properties on the Popup menu.
D In the Workbench right pane:	
Either	Or
Double click Event Editor .	<ol style="list-style-type: none"> a. Right-click Event Editor. b. Select Properties on the Popup menu.

E	Press Alt+Enter on the keyboard.
---	----------------------------------

4. Right-click **Event Editor**.
5. Select Properties on the Popup menu.
6. Right-click **Event Editor**.
7. Select Properties on the Popup menu.

Step 2. Review Event Editor Features

Step 2. Review Event Editor Features

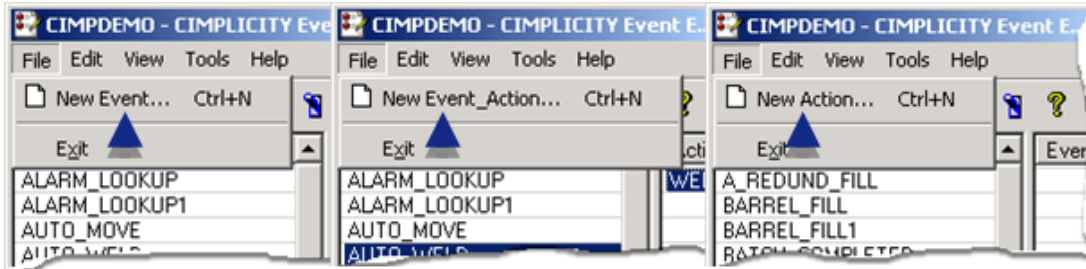
Option 2.1 <i>(on page 35)</i>	Event Editor menus.
Option 2.2 <i>(on page 40)</i>	Event Editor toolbar.
Option 2.3 <i>(on page 41)</i>	Event Editor shortcut keys.

Option 2.1. Event Editor Menus

You can use the menu options to create new events and actions, modify, delete or copy selected events and actions, reorder the actions for an event, display the attributes for an event or action, toggle dynamic updates, and access Help.

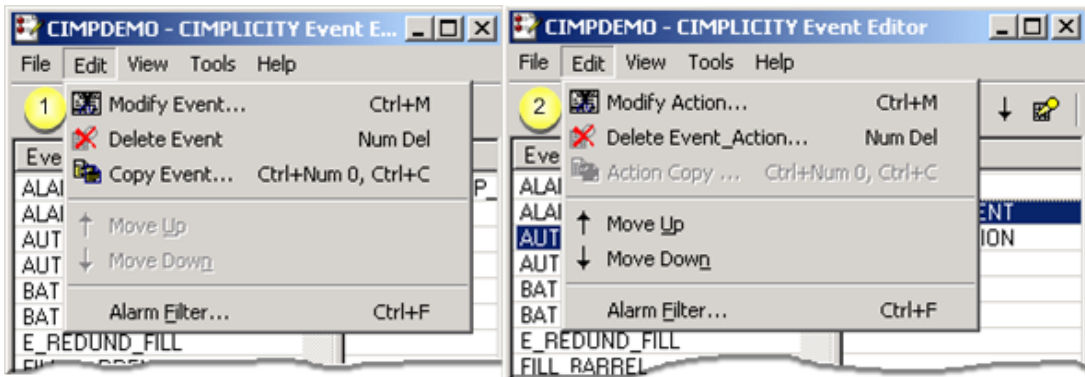
- File menu
- Edit menu
- View menu
- Tools menu
- Help menu

File menu



Option	Select- ed Pane	View	Description
New Event...	Left	Event (on page 42)	Creates a new Event. This option is displayed if the Event pane is active.
New Event_ Action...	Right	Event (on page 74)	Creates a new action for the currently selected Event. This option is displayed if the Event pane is active, and you have clicked the mouse once in the Action pane.
New Ac- tion...	Left	Ac- tion (on page 60)	Creates a new Action. This option is displayed if the Action pane is active.
Exit			Exits the Event Editor.

Edit Menu



1	An event is selected.
2	An action is selected.

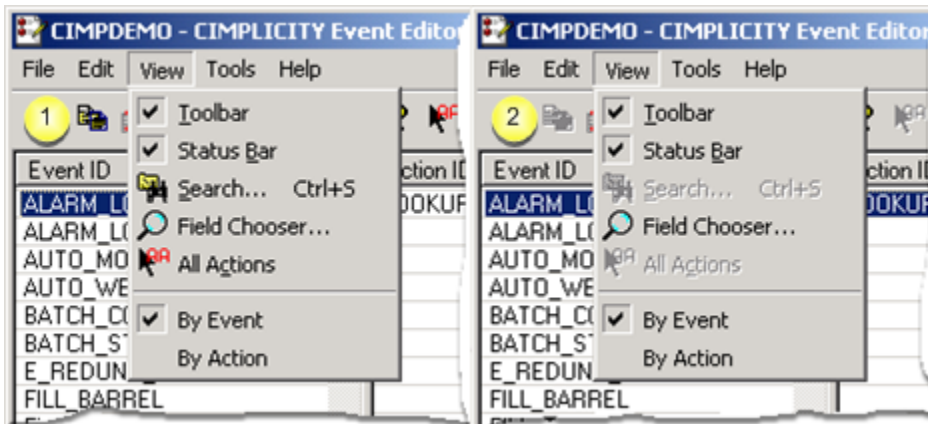
Modify Event	Opens the Modify Event dialog box, and lets you change the Event Type and associated fields.
Modify Action	Opens the Modify Action dialog box, and lets you change the Action Type and associated fields.
Delete Event	Deletes the selected Event from the list of available Events
Delete Event-Action	Removes the selected Action from the list of Actions for the selected Event.
Delete Action	Deletes the Action. This function will remove the Action from all Events that use it and remove it from the list of available Actions.
Copy Event	Copies the selected Event to a new Event. You can also choose to copy the Actions.
Move Up	While viewing Event-Actions, controls the execution order of the selected Action by moving it up in the list of Actions for the Event.
Move Down	While viewing Event-Actions, controls the execution order of the selected Action by moving it down in the list of Actions for the Event.
Alarm Filter	Opens the Alarm Setup dialog box and lets you set the filter for the alarms the Event Manager will respond to.



Note:

Scripts run a-synchronously, so their order in the list does not guarantee their order of execution. Other actions, like **Setpoint**, can be ordered.

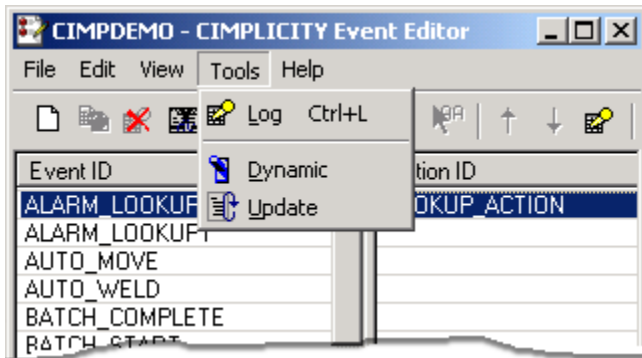
View menu



1	An event is selected.
2	An action is selected.

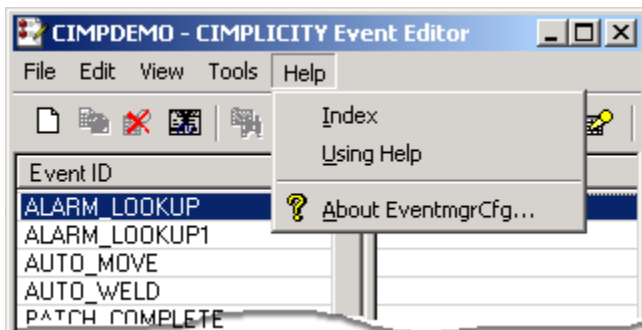
Toolbar	Toggles the display of the Toolbar.
Status Bar	Toggles the display of the Status Bar.
Search	If you are displaying By Event , opens the Event Search dialog box. If you are displaying By Action , opens the Action Search dialog box.
Event At-tributes...	If you are displaying By Event , opens the Configure Display Attributes dialog box for Events, and lets you select Event attributes to display in the window.
Action At-tributes...	If you are displaying By Action , opens the Configure Display Attributes dialog box for Ac-tions, and lets you select Action attributes to display in the window.
All Ac-tions	Displays all Actions in the Action pane. You can then select Actions and drag them into an Event.
By Event	Displays Event and Action information by Event.
By Action	Displays Event and Action information by Action.

Tools menu



Log	Enable or disables logging of Events and Actions.
Dynamic	Enables or disables Dynamic Configuration of points, alarms, etc., when configuring Events or Actions.
Update	Dynamically updates the Basic Control Engine with the current Event configuration and scripts used by the Actions in the configuration. The Basic Control Engine normally loads and compiles your scripts at project startup. If you modify a script and save it to disk while your project is running, the Basic Control Engine will not load the modified script until you perform an Update or the until project is stopped and restarted.

Help menu



Index	Displays the main Help window for the Event Editor.
Using Help	Displays the main Help window for Windows operating system.
About Eventmgr Cfg...	Displays the program identification, version number, and copyright for the Event Editor.

Option 2.2. Event Editor Toolbar

1. Click View on the Event Editor menu bar.
2. Do one of the following.
 - Check Toolbar to display the toolbar.
 - Clear Toolbar to hide the toolbar.

The buttons on the Tools toolbar are as follows.



1. [#unique_10_Connect_42_New \(on page 40\)](#)
2. [#unique_10_Connect_42_Copy \(on page 40\)](#)
3. [#unique_10_Connect_42_Delete \(on page 40\)](#)
4. [#unique_10_Connect_42_Modify \(on page 40\)](#)
5. [#unique_10_Connect_42_Search \(on page 40\)](#)
6. [#unique_10_Connect_42_Attributes \(on page 41\)](#)
7. [#unique_10_Connect_42_Dynamic \(on page 41\)](#)
8. [#unique_10_Connect_42_About \(on page 41\)](#)
9. [#unique_10_Connect_42_Show \(on page 41\)](#)
10. [#unique_10_Connect_42_ActionOrderUp \(on page 41\)](#)
11. [#unique_10_Connect_42_ActionOrderDown \(on page 41\)](#)
12. [#unique_10_Connect_42_ToggleLog \(on page 41\)](#)
13. [#unique_10_Connect_42_Update \(on page 41\)](#)

1	Event is selected	
2	Action is selected	
A	New	Creates a new Event or Action record.
B	Copy	Makes a copy of the selected event or action.
C	Delete	Deletes the selected event(s) or action(s)
D	Modify	Modifies the selected event or action
E	Search	Searches for specified events or actions.

F	Attributes	Opens the Field Chooser dialog box for events or actions.
G	Dynamic	Enables/disables dynamic configuration updates.
H	About	Displays program information, version number, and copyright.
I	Show all actions	Shows all actions.
J	Action order up	Moves the selected action up in the list for an event.
K	Action order down	Moves the selected action down in the list for an event.
L	Toggle Logging	Enables/disables event action logging.
M	Update	Updates Control Manager runtime.

Option 2.3. Event Editor Shortcut Keys

The following are the more commonly used keystrokes that are available for your use in the Event Editor:

Key-stroke	Description
Ctrl+N	Creates a new Event, Event-Action, or Action.
Ctrl+M	Modifies an Event or Action.
Del	Deletes an Event or Action.
Ctrl+C	Copies an Event or Action.
Ctrl+S	Searches for selected Events or Actions.
Ctrl+L	Toggles logging for Events and Actions.
F1	Opens the Help window for the Event Editor.
Ctrl+F	Opens the Alarm Setup dialog box.

Step 3. Configure an Event

Step 3. Configure an Event

<p>Step 3.1 <i>(on page 42)</i></p>	<p>Create an event.</p>
<p>Step 3.2 <i>(on page 58)</i></p>	<p>Enter advanced event specifications.</p>

Step 3.1. Create an Event

Step 3.1. Create an Event

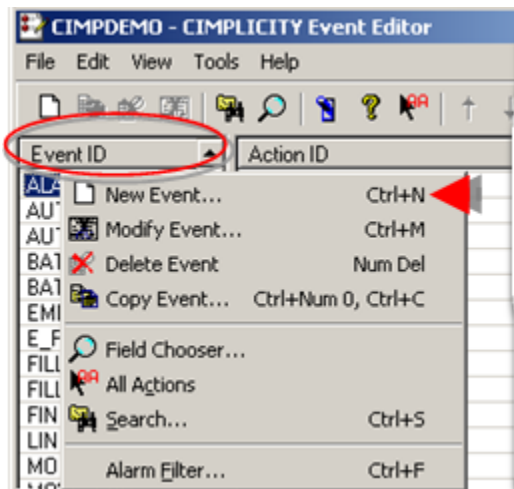
1. Click View on the CIMPLICITY Event Editor menu bar.
2. Select [By Event \(on page 38\)](#).
3. Do one of the following.

Method 1

Click the New button  on the Event Editor toolbar.

Method 2

- a. Right-click the Event Editor left pane.
- b. Select New Event on the popup menu.



Method 3

Select New Event on the Event Editor [File menu \(on page 36\)](#).

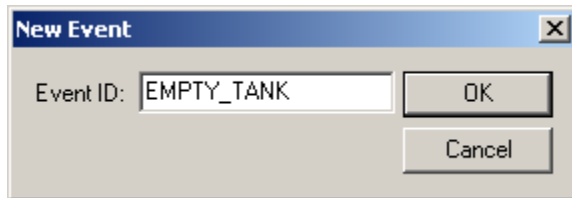
Method 4

Press Ctrl+N on the keyboard.

A New Event dialog box opens.

4. Enter a name in the **Event ID** field.

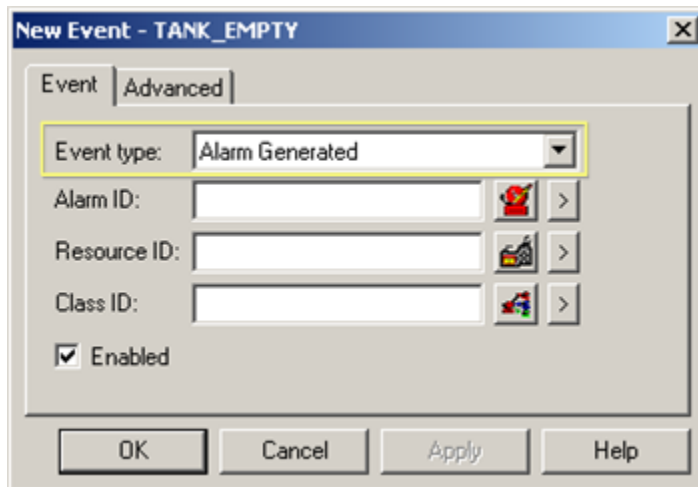
Note: The event ID can be a maximum of **256** characters and mixed case.



5. Click OK.

An expanded New Event dialog box opens.

6. Select an Event in the **Event Type** field.



7. Configure the event you select.

Events are:

- Alarm Acknowledged
- Alarm Deleted
- Alarm Generated
- Alarm Reset
- Point Change

- Point Equals
- Point Transition High
- Point Transition Low
- Point Unavailable
- Point Update
- Run Once
- Timed



Note:

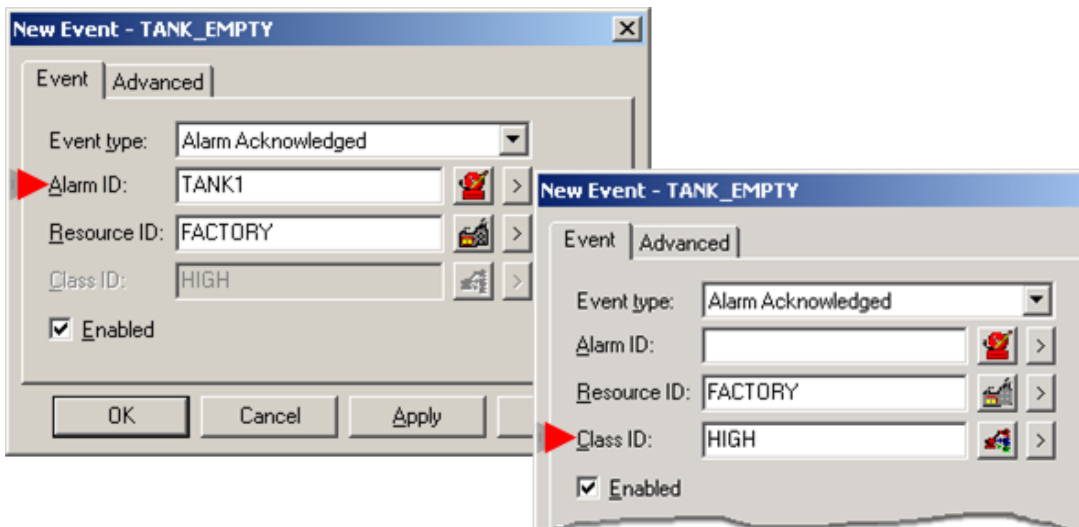
You can modify these fields in the [Modify Event \(on page 76\)](#) dialog box.


The dialog box closes and the new event appears in the Event list in the CIMPLICITY Event Editor window.






Alarm Acknowledged Events

An **Alarm Acknowledged Event** occurs when the alarm identified in the **Alarm ID** field for the Event is acknowledged.

Fields are as follows.



Field	Description
Alarm ID	ID of an alarm or wild card to specify a group of alarms that will trigger this event when the alarm is acknowledged.
	 Opens the Alarm browser.

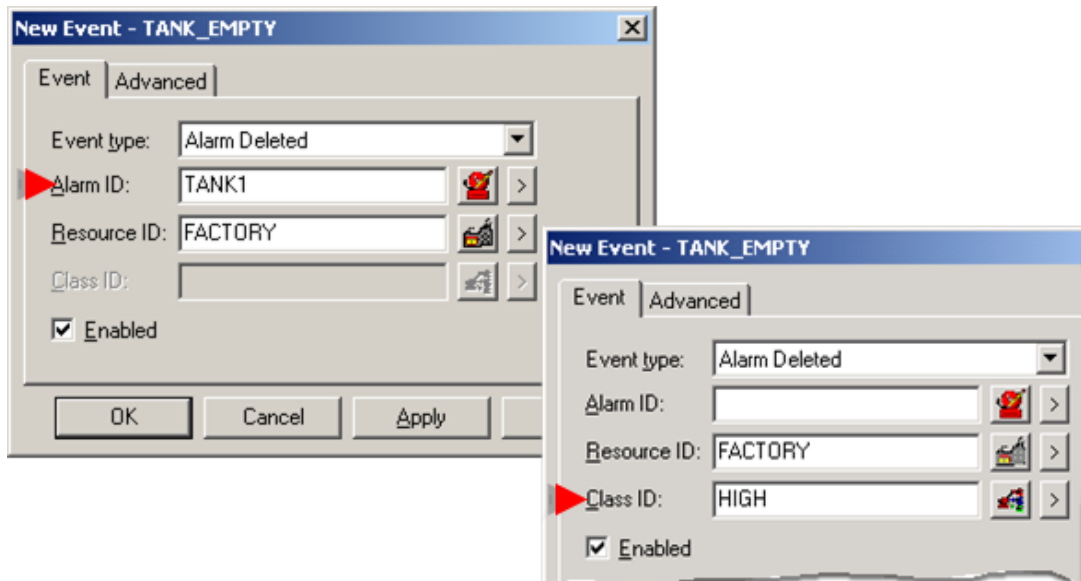
		Displays popup menu to create a new alarm, browse for or edit an existing alarm
Re-source	With:	The event will be generated:
	No entry	Whenever the alarm is acknowledged.
	An entry	When the alarm is acknowledged for that resource
		Opens the Resource browser.
		Displays popup menu to create a new resource, browse for or edit an existing resource.
Class ID	Alarm classification that will evaluate this event. Note: This field is unavailable if an Alarm ID is selected	
		Opens an Alarm Class browser.
		Displays popup menu to create a new alarm class, browse for or edit an existing alarm class.
Enabled	Checked	Enables the event.
	Clear	Disables the event.







**Note:**

Alarms can be acknowledged manually by operators, or automatically via software.

Alarm Deleted Events

An **Alarm Deleted** Event occurs when the alarm identified in the **Alarm ID** field for the Event is deleted.



Field	Description	
Alarm ID	ID of an alarm or wild card to specify a group of alarms that will trigger this event when the alarm is deleted.	
		Opens the Alarm browser.
		Displays popup menu to create a new alarm, browse for or edit an existing alarm
Re-source	With:	The event will be generated:
	No entry	Whenever the alarm is acknowledged.
	An entry	When the alarm is acknowledged for that resource
		Opens the Resource browser.
		Displays popup menu to create a new resource, browse for or edit an existing resource.
Class ID	Alarm classification that will evaluate this event. Note: This field is unavailable if an Alarm ID is selected	
		Opens an Alarm Class browser.
		Displays popup menu to create a new alarm class, browse for or edit an existing alarm class.

En-abled	Checked	Enables the event.
	Clear	Disables the event.

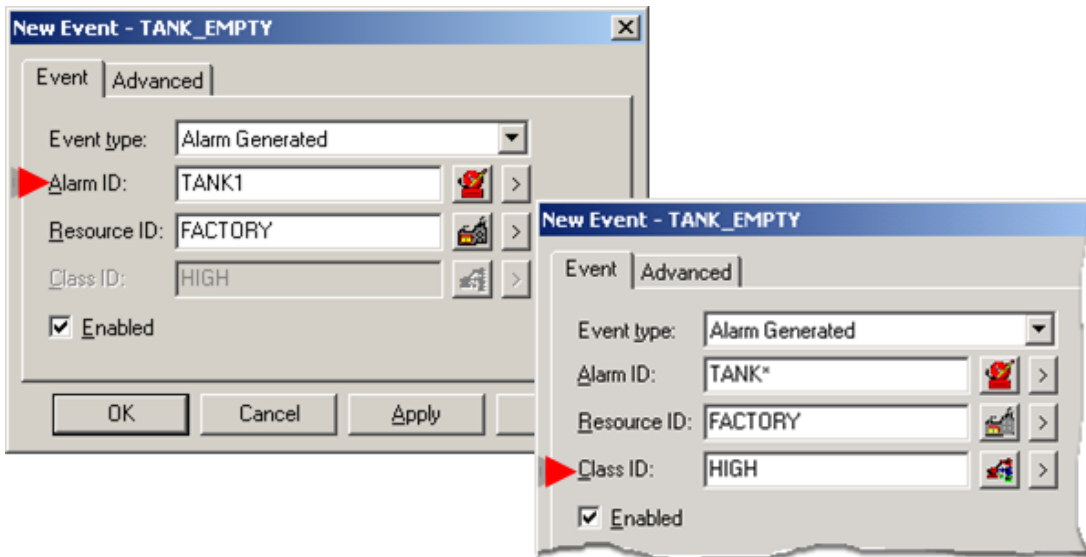




Note:





Alarms may be deleted manually by operators, or automatically via software.

Alarm Generated Events

An **Alarm Generated** Event occurs when the alarm identified in the **Alarm ID** field for the Event is generated.



Field	Description	
Alarm ID	ID of an alarm or wild card to specify a group of alarms that will trigger this event when the alarm is generated.	
		Opens the Alarm browser.
		Displays popup menu to create a new alarm, browse for or edit an existing alarm
Re-source	With:	The event will be generated:
	No entry	Whenever the alarm is acknowledged.

	An entry	When the alarm is acknowledged for that resource
		Opens the Resource browser.
		Displays popup menu to create a new resource, browse for or edit an existing resource.
Class ID	Alarm classification that will evaluate this event. Note: This field is unavailable if an Alarm ID is selected	
		Opens an Alarm Class browser.
		Displays popup menu to create a new alarm class, browse for or edit an existing alarm class.
Enabled	Checked	Enables the event.
	Clear	Disables the event.

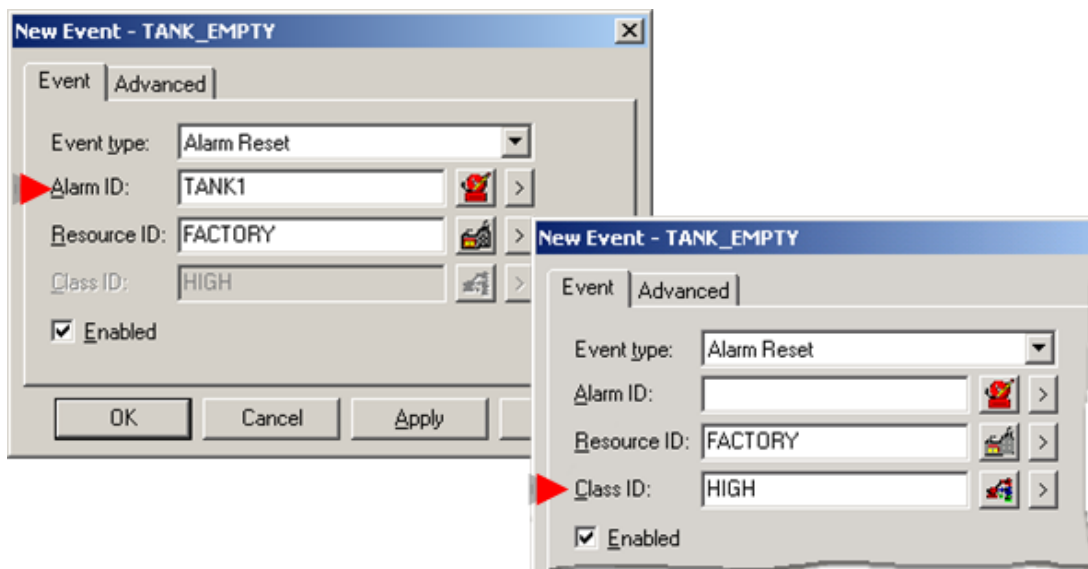








Note:

All alarm events allow wild cards for pattern matching. Valid wild cards are * and ?. In the above example, the event "Alarm" will occur whenever a HIGH Class alarm occurs.

Alarm Reset Events

An **Alarm Reset** Event occurs when the alarm identified in the **Alarm ID** field for the Event is reset.



Field	Description	
Alarm ID	ID of an alarm or wild card to specify a group of alarms that will trigger this event when the alarm is reset.	
		Opens the Alarm browser.
		Displays Popup menu to create a new alarm, browse for or edit an existing alarm
Re-source	With:	The event will be generated:
	No entry	Whenever the alarm is acknowledged.
	An entry	When the alarm is acknowledged for that resource
		Opens the Resource browser.
		Displays Popup menu to create a new resource, browse for or edit an existing resource.
Class ID	Alarm classification that will evaluate this event. Note: This field is unavailable if an Alarm ID is selected	
		Opens an Alarm Class browser.
		Displays Popup menu to create a new alarm class, browse for or edit an existing alarm class.
Enabled	Checked	Enables the event.
	Clear	Disables the event.

**Note:**



Alarms can be reset manually by operators, or automatically via software.

Point Change Events

A **Point Change** Event occurs when value of the point identified in the **Point ID** changes.



**Note:**

Point value changes to and from the unavailable value are not Point Change events. Use the [Point Update \(on page 54\)](#) event to detect these changes.

Field	Description	
Point ID	ID of a point that will trigger this event when the point value changes.	
		Opens the Point browser.
		Displays Popup menu to create a new alarm, browse for or edit an existing alarm
Enabled	Checked	Enables the event.
	Clear	Disables the event.

Point Equals Events

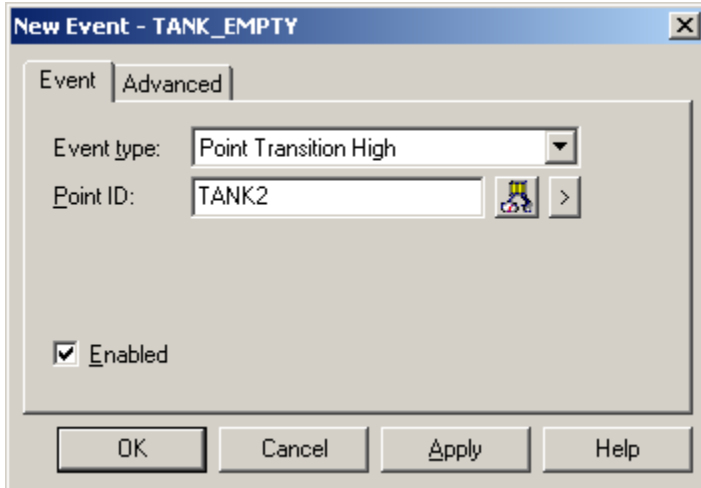
A **Point Equals** Event occurs when value of the point identified in the **Point ID** field equals the value in the **Value** field.



Field	Description	
Point ID	ID of a point that will trigger this event when the value equals the value in the Value field.	
		Opens the Point browser.
		Displays Popup menu to create a new point, browse for or edit an existing point.
Value	Value that will trigger the event.	
Enabled	Checked	Enables the event.
	Clear	Disables the event.

Point Transition High Events

A **Point Transition High Event** occurs when value of the Digital type point identified in the **Point ID** field transitions to HIGH (that is, it changes value from 0 to 1).

The code explicitly runs the action for transition high (or transition low events) if the value was unavailable.

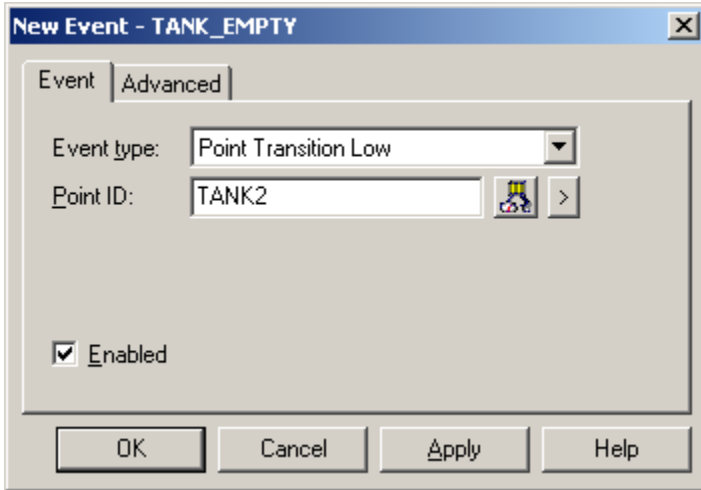




Field	Description	
Point ID	ID of a point that will trigger this event when the point value transitions to HIGH. If the point is an array point, you can specify the element that will trigger this event. To specify an element, append the index in brackets at the end of the Point ID (for example, ARRAY_PT[3]). If you do not specify the element for an array point, the first element is assumed.	
		Opens the Point browser.
		Displays Popup menu to create a new point, browse for or edit an existing point.
Enabled	Checked	Enables the event.
	Clear	Disables the event.

Point Transition Low Events

A **Point Transition Low** Event occurs when value of the Digital type point identified in the **Point ID** field transitions to LOW (that is, it changes value from 1 to 0).

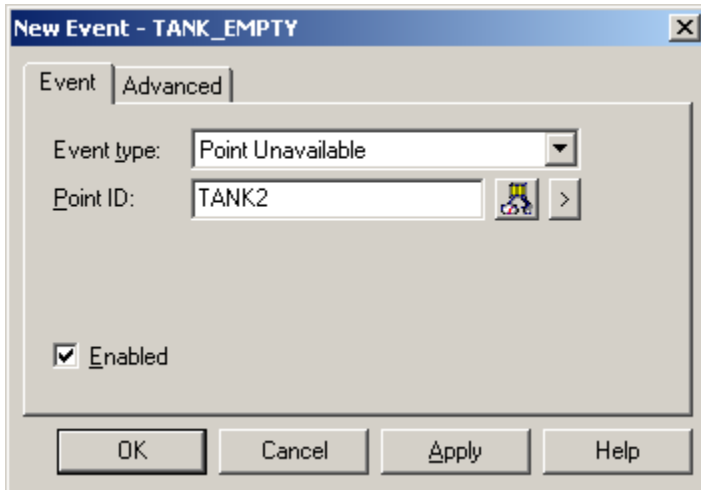
The code explicitly runs the action (for transition high or) transition low events if the value was unavailable.





Field	Description	
Point ID	ID of a point that will trigger this event when the point value transitions to LOW. If the point is an array point, you can specify the element that will trigger this event. To specify an element, append the index in brackets at the end of the Point ID (for example, ARRAY_PT[3]). If you do not specify the element for an array point, the first element is assumed.	
		Opens the Point browser.
		Displays Popup menu to create a new point, browse for or edit an existing point.
Enabled	Checked	Enables the event.
	Clear	Disables the event.

Point Unavailable Events

A **Point Unavailable Event** occurs when value of the point identified in the **Point ID** field becomes unavailable.



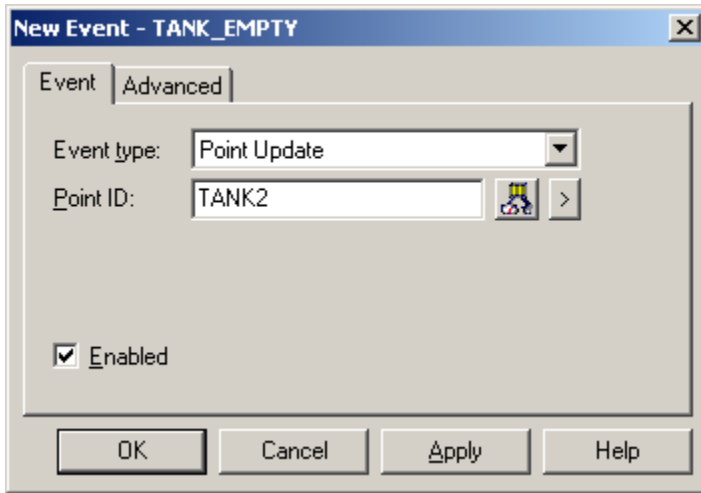
Field	Description	
Point ID	ID of a point that will trigger this event when the point becomes unavailable.	
		Opens the Point browser.
		Displays Popup menu to create a new point, browse for or edit an existing point.
Value	Value that will trigger the event.	
Enabled	Checked	Enables the event.
	Clear	Disables the event.



Point Update Events

A **Point Update** Event occurs when value of the point identified in the **Point ID** field is updated. The rate at which the point is updated is a function of its **Update criteria**, which will be one of the following:

Update Criteria	The point is updated:
On Scan	At each scan interval.
On Change	When its value changes.
On Demand	On request by a CIMPLICITY process.

On Demand On Scan	The point is updated at each scan interval while it is being requested by a CIMPLICITY process.
On Demand On Change	When its value changes while it is being requested by a CIMPLICITY process.
Poll Once	When the point is polled, which is once at startup.
Unsolicited	Whenever the device determines that an update is needed.



Field	Description	
Point ID	ID of a point that will trigger this event when the point value updates.	
		Opens the Point browser.
		Displays Popup menu to create a new point, browse for or edit an existing point.
Enabled	Checked	Enables the event.
	Clear	Disables the event.

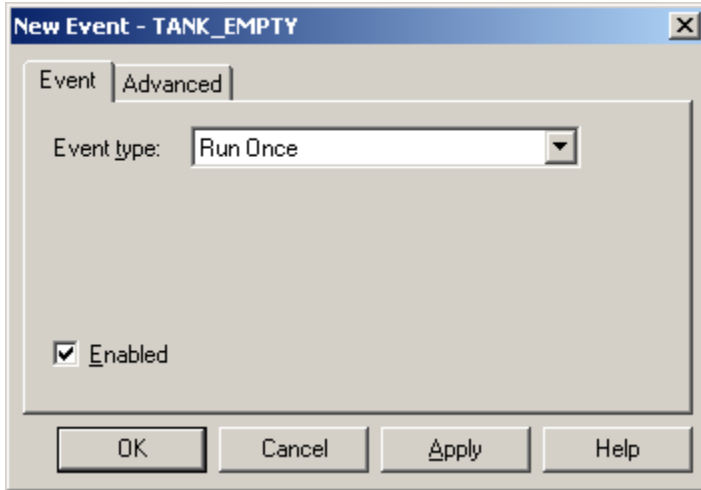


Note:

Point value changes to and from the unavailable value are also [Point Update \(on page 54\)](#) events.

Run Once

The Event Type, **Run Once**, is invoked once when the Event Manager starts.



Field	Description	
Enabled	Checked	Enables the event.
	Clear	Disables the event.

Timed Events

This topic describes about Timed Events.

A **Timed Event** occurs when the time identified in the **Event Time** field occurs.

New Event - myevent01 ✕

Event **Advanced**

Event type: Timed ▼

Event time: 12 : 0 : 0 AM ▼ HH:MM:SS

Event int.: 0 : 0 : 0 HH:MM:SS

Enabled

OK
Cancel
Apply
Help

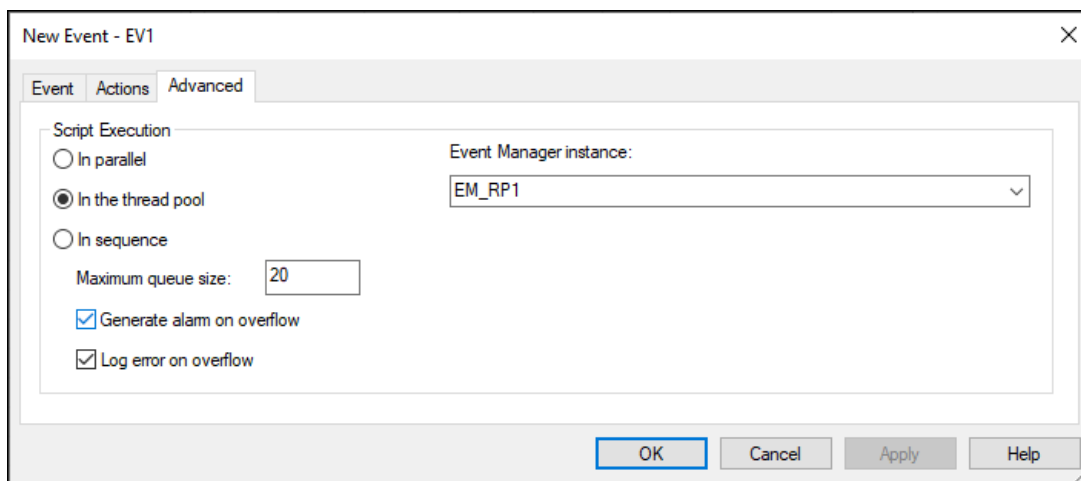
Field	Description
Event type	The type of the event. In this case, it is Timed Event.
Event time	Time when the Event is to be triggered. Format HH:MM:SS AM or PM
Event int	Interval of time after the Event Time when the Event will be rescheduled. If you do not want to reschedule the Event, leave zeros in these fields.
Enabled	Allows you to enable or disable the event.

Examples

Schedule an event for:	You can do the following:
Every hour on the quarter hour	1. Enter 12:15:00 AM in the Event Time field. 2. Enter 01:00:00 in the Event Int field. Result: The event is scheduled at 12:15:00 AM, 01:15:00 AM, 02:15:00 AM, etc.
Every 15 minutes	1. Enter 12:00:00 AM in the Event Time field. 2. Enter 00:15:00 in the Event Int field. Result: The event is scheduled at 12:15:00 AM, 12:30:00 AM, 12:45:00 AM, etc.
2:30 AM every day	1. Enter 02:30:00 AM in the Event Time field. 2. Enter 00:00:00 in the Event Int field. Result: The event is scheduled at 2:30 AM everyday.

Step 3.2. Enter Advanced Event Specifications

The various options in the Advanced section in the New Event dialog box are as follows.



Option	Description
Script Execution - In Parallel	Runs a script each time an event is invoked. More than one copy of the script may run at a time. You must use critical sections to control access to resources.


cu- tion		The maximum number of scripts that run in parallel is undefined. Thus, several threads are created to execute the scripts in parallel, thereby requiring more computing resources.
	In the thread pool	Runs the script in threads from a thread pool. The thread pool is created when the EMRP process starts. The scripts also run in parallel, but the number of threads is limited to the size of the thread pool. For details on the thread pool size and how to configure it, see Running a script in parallel (in the thread pool) .
	In Se- quence	(Default) When an event is triggered, if an existing instance of the event is still executing, the script will be queued to start after the current script is done. The maximum number of script actions that can run simultaneously is CE_MAX_THREADS + CE_POOL_THREADS.
Max- i- mum Queue		If the option In sequence is selected, you must specify a maximum queue size. In this case, when more than 20 events are queued, the oldest will be discarded.
Gen- erate Alarm on Over- flow		(Default) If the sequential queue overflows, select this check box to generate an \$SEM_QUEUE alarm. If your event is an alarm event, generating an alarm may cause your event to trigger again and generate another alarm. This will cause a circular cycle of alarms.
Log Error on Over- flow		(Default) If the sequential queue overflows, check to generate a message in the status log.
Event Man- ag- er in- stance		Allows you to select an Event Manager instances in a project and associate it to an class event. This allows logical separation of critical/key events into their own instances which can prevent unwanted interactions. For more information on Event Manager Instance, refer to About Multiple Event Manager Resident Process (EMRP) (on page 97) . Once you created a class event and associated it with an Event Manager instance, you must check if the instance is running (on page 104) , and you can also test the event with the Basic Control Engine UI (BCEUI) (on page 106) .

Step 4. Create an Action

Step 4. Create an Action

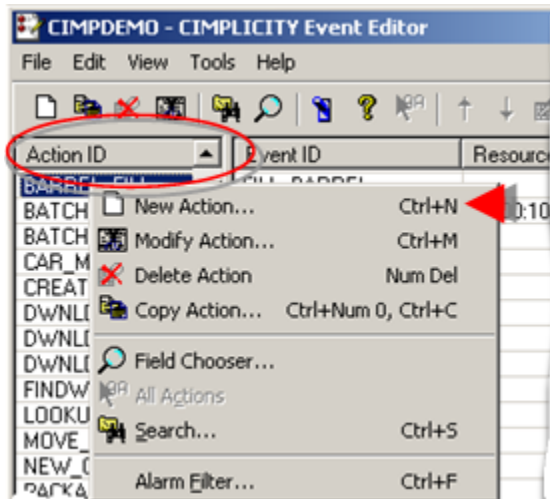
1. Click View on the Event Editor menu bar.
2. Select [by Action \(on page 38\)](#).
3. Do one of the following.

Method 1

Click the New  button on the Event Editor toolbar.

Method 2

- a. Right-click the Event Editor left pane.
- b. Select New Action on the popup menu.



Method 3

Select New Action on the Event Editor [File menu \(on page 36\)](#).

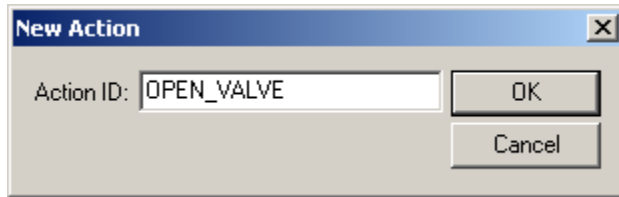
Method 4

Press Ctrl+N on the keyboard.

A New Action dialog box opens.

4. Enter a name in the **Action ID** field.

Note: The action ID can be a maximum of **256** characters and mixed case.

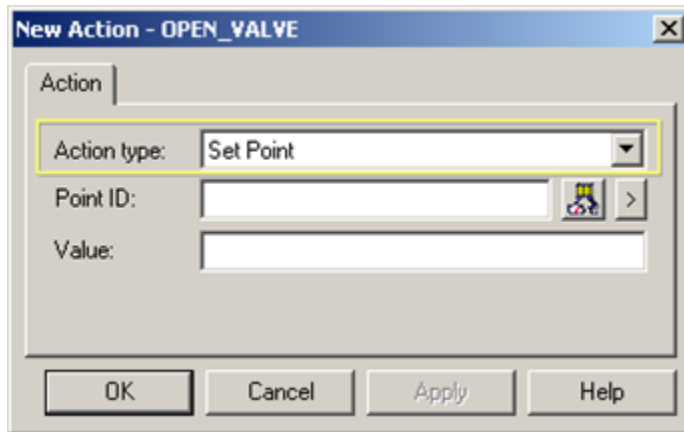
**Important:**

The name must begin with a letter, not a number.

5. Enter the name of the new Action in the **Action ID** field and click **OK**.

An expanded New Action dialog box opens.

6. Select an action in the **Action type** field.



7. Configure the action you select.

Alarm Look-Up
Log Only
Point Alarm Acknowledge
Point Alarm Disable
Point Alarm Enable
Recipe Upload/Download
Run Script
Set Point

Source Transition Set
Transition Set

The dialog box closes and the new action appears in the Action list in the CIMPLICITY Event Editor window.



Note:

You can modify these fields in the [Modify Action \(on page 77\)](#) dialog box.

Alarm Look-Up Actions

(Required) enter the name of the CIMPLICITY Alarm ID for which the action will be taken.



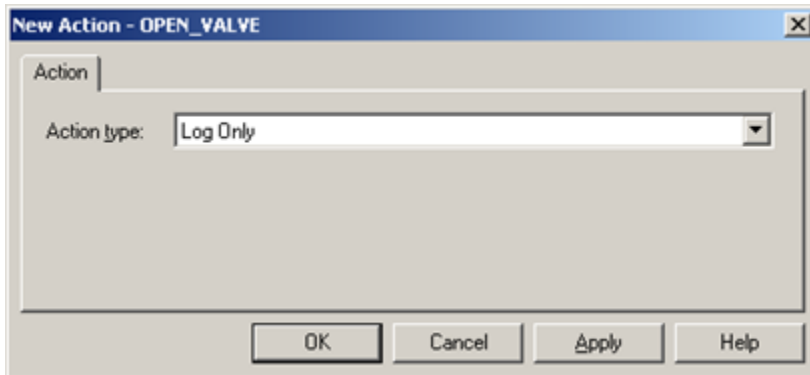
Important:

When you create the Alarm ID in the Alarm Definition dialog box, you must:

1. Select `$_CIMBASIC` in the **Alarm type** field.
2. Enter one `%s` parameter in the **Alarm message** field to hold the Alarm Message defined for the Point Value.

Log Only Actions

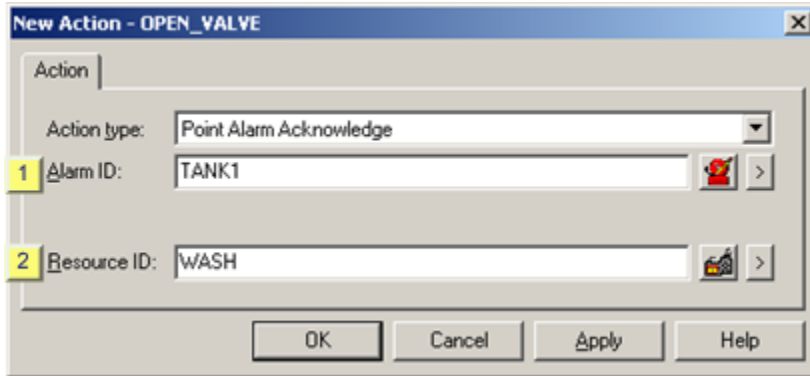
A **Log Only** action logs the associated Event in the Database Logger Event Log. No other action is taken.



Point Alarm Acknowledge Actions

A **Point Alarm Acknowledge** action acknowledges the alarm defined by the **Alarm ID** and **Resource ID**.

To create this Action, enter the following information in the New Action dialog box:



1. #unique_33_Connect_42_i2Resource (on page 63)
2. #unique_33_Connect_42_i1AlarmID (on page 63)

1 (on page 63)	Alarm ID
2 (on page 63)	Resource

1	Alarm ID
---	----------

ID of an alarm to be acknowledged.

(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Alarm browser.
	Popup Menu button	Displays Popup menu to create a new alarm, browse for or edit an existing alarm

2	Resource
---	----------

Resource of the alarm to be acknowledged.



Note:

This field is automatically filled in, when an Alarm ID is entered, with the resource assigned to the alarm.

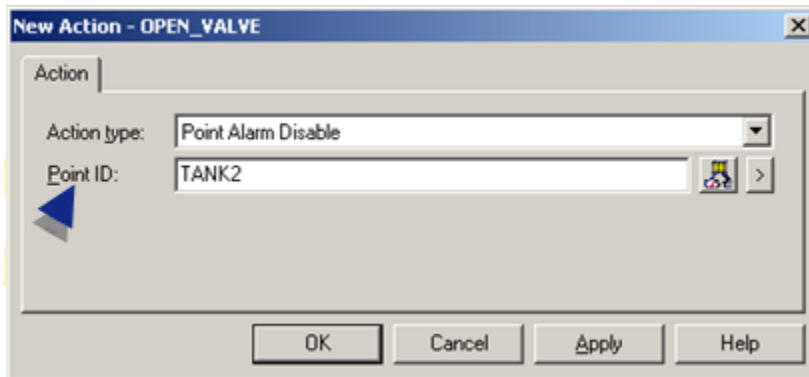
(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Resource browser.
	Popup Menu button	Displays Popup menu to create a new resource, browse for or edit an existing resource.

Point Alarm Disable Actions

A **Point Alarm Disable** action disables alarming for the point in the **Point ID** field.

To create this Action, enter the following information in the New Action dialog box:



Point ID

ID of a point for which alarming is to be disabled.

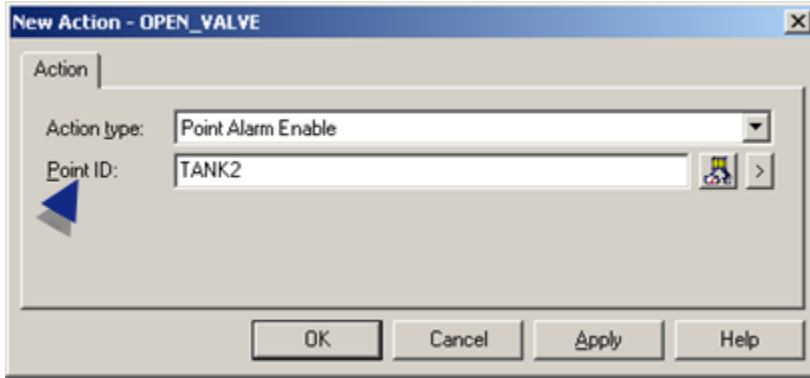
(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Point browser.
	Popup Menu button	Displays Popup menu to create a new point, browse for or edit an existing point.

PointAlarm Enable Actions

A Point Alarm Enable action enables alarming for the point in the **Point ID** field.

To create this Action, enter the following information in the New Action dialog box:



Point ID

ID of a point for which alarming is to be enabled.

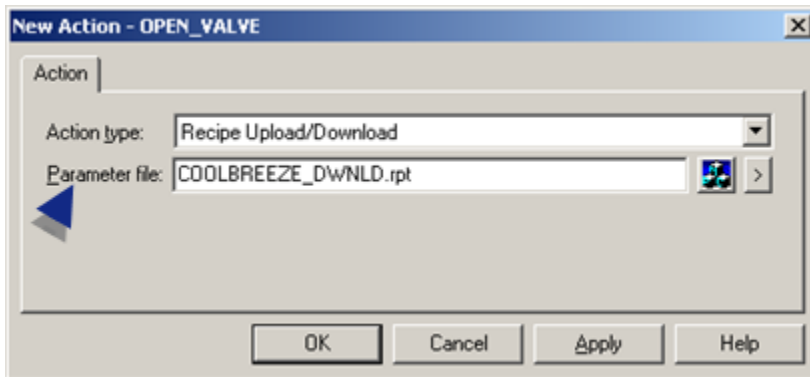
(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Point browser.
	Popup Menu button	Displays Popup menu to create a new point, browse for or edit an existing point.

Recipe Upload/Download

A Recipe Upload/Download action uploads or downloads the recipe defined by a selected parameter file.

To create this Action, enter the following information in the New Action dialog box:



Parameter
file

Automatic actionfile that was created in Recipes.

(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Select a Recipe File browser.
	Popup Menu button	Displays a Popup menu to create a new automatic action Recipe file, browse for or edit an existing automatic action Recipe file.

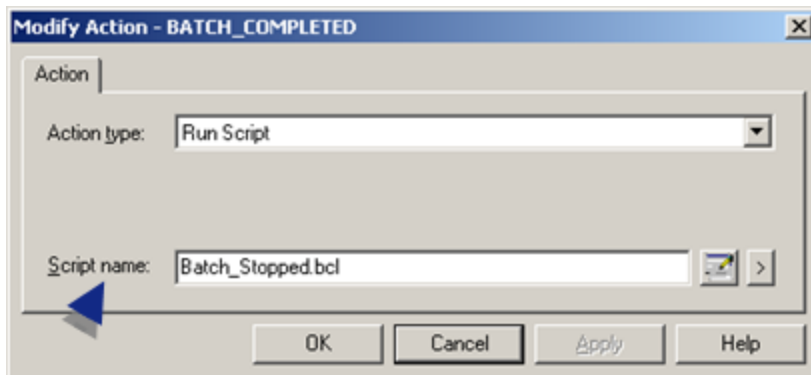
The selected script name cannot exceed 48 characters. If you try to select an action with a name longer than 48 characters CIMPLICITY will not allow you to use it.

Run Script Actions


A Run Script action executes a selected script. Event Manager supports the following types of scripts:

- Basic Script
- C# Script
- Visual Basic .Net Script
- Python Script

The script is run in parallel with all actions that are being executed for the event. In other words, the Basic Control Engine does not wait for the script to complete before it initiates the next action defined for the event.




To add an existing script:

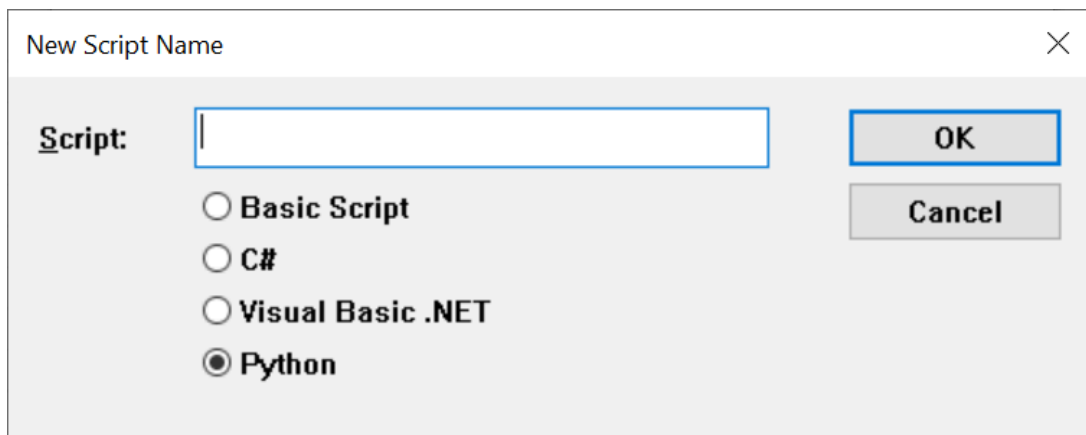
1. Select the Browse button .
2. Select the script you want to add to the action.



3. Select **OK**.

To add a new script:

1. Select the **Popup Menu** button .
2. Select **New**.
3. Select the Script type.



4. Enter a name for the script in **Script** text box.
5. Select **OK**. The corresponding script editor opens.

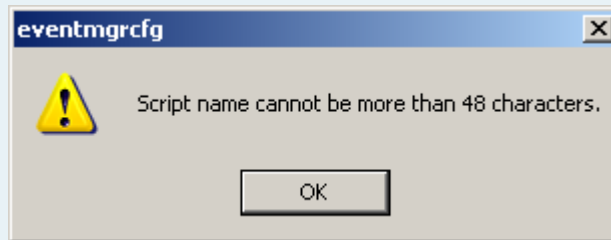
6. Edit and save the script.

When the event to which the script is added occurs, the script gets executed. You can view the status of the script execution in the BCE User Interface.



Note:

The selected script name cannot have more than 48 characters. If you try to select a script with a name longer than 48 characters CIMPLICITY will display an error message and will not allow you



to use it.



Important:

The Basic Control Engine loads and compiles your scripts when your project starts up. If you modify a script and save it to disk while your project is running you need to do either of the following to make the Basic Control Engine load the modified script.

Method 1

Click Tools on the Event Editor menu bar.

Select Update.

Method 2

Stop the project.

Restart the project.

Types of Script Execution

When a configured event with a script action is triggered, the script can be executed in the following ways:

- In sequence
- In parallel (includes thread pool)

To configure an event with the type of script execution, see [Step 3.2. Enter Advanced Event Specifications \(on page 58\)](#).

Running a script in parallel vs. in sequence

You can run scripts in parallel if they wait on Input/Output (I/O) operations for extended periods of time. This will support running more threads.

You can run scripts in sequence if they interact with an external system that cannot perform multiple operations in parallel.

The set of threads used to run scripts in parallel or in sequence are managed by a common thread manager. The `CE_MAX_THREADS` global parameter controls the maximum number of threads the thread manager will use to run scripts, and decides when and if the script will be run.

- If there are fewer than `CE_MAX_THREADS` scripts currently running in parallel, the script will be run immediately.
- If there are `CE_MAX_THREADS` or more scripts running in parallel, the script is discarded and a Too many executing threads, action ignored message is logged to the status log.
- If there is another script running in sequence for a configured event and there are fewer actions queued than the maximum queue size of the configured event, the script is queued.
- If there is no other script running in sequence for a configured event and there are fewer than `CE_MAX_THREADS` scripts currently running in sequence, the script is run immediately.
- If as many actions as the maximum queue size of the configured event are queued, the script running in sequence is discarded and an alarm is generated and/or a message is logged to the status log depending on the configuration of the event.
- If there is no other script running in sequence for a configured event and there are `CE_MAX_THREADS` or more scripts currently running in sequence, the script is discarded and a Too many executing threads, action ignored message is logged to the status log.

Running a script in parallel (in the thread pool)

You can run a CPU-intensive script in parallel in a set of threads managed by a thread pool. The thread pool should be sized so that there is one thread per logical processor in the system. This helps minimize the time spent in switching CPU cores.

**Note:**

For cores that support hyperthreading, the number of logical processors is twice the number of cores. For cores that do not support hyperthreading, the number of logical processors is equal to the number of cores.

The `CE_POOL_THREADS` global parameter controls the maximum size of the thread pool, and also decides when the script will be run.

- If there are fewer than `CE_POOL_THREADS` scripts currently running, the script will be run immediately.
- If there are `CE_POOL_THREADS` or more scripts running, the script is queued.

To configure the `CE_MAX_THREADS` and `CE_POOL_THREADS` global parameters:

- Select Project, and then select Properties.
- In the Settings tab of the Project Properties dialog box, select Event Editor, and then select Settings.
- In the Setup dialog box, select the Set thread pool size to option, and enter a number.

Notes

- The thread pool size ranges between 0 and 200, and when calculated automatically will be twice the number of logical processors in the system. When the size is set to 0, its size is automatically calculated.
- `CE_MAX_THREADS` should be set to the expected number of simultaneous events. The actions of the surplus events triggered will be ignored.
- The maximum number of script actions that can run simultaneously is `CE_MAX_THREADS + CE_POOL_THREADS`.
- BCL and .NET scripts share the same set of threads.
- When a script is started, it can run in any available thread.
- The `CE_THREAD_TIMEOUT` global parameter controls the number of seconds a thread managed for parallel and sequential events will be idle before it is freed. This period should be long enough so that regularly executed events do not need to create threads, but short enough so that infrequent events do not cause the event manager to consume an abnormal amount of memory for extended periods of time.
- The `CE_MIN_STANDBY_THREAD_COUNT` global parameter controls the number of threads allowed by the event manager to be idle indefinitely. Threads that are idle for `CE_THREAD_TIMEOUT` seconds will not be freed if there are `CE_MIN_STANDBY_THREAD_COUNT` or fewer threads in the idle state.

Variable scope and lifetime

In BCL, you can declare public or private variables at the module level, outside of any function. In .NET, you can declare static or instance variables at the class level, outside of any function.

The following table provides the differences between BCL and .NET variables:

Variable	Scope of variable	Lifetime of variable value	Shared	Multi-threading issues
BCL public	Global, visible to all script files (modules)	Forever, across event instances	Yes	Yes
BCL private	Visible only to this script file (module)	Forever, across event instances	Yes	Yes
.NET class static	Visible only to this script file (AppDomain)	Forever, across event instances	Yes	Yes
.NET class instance	Visible only to this script file (AppDomain)	Forever, across event instances	No	No

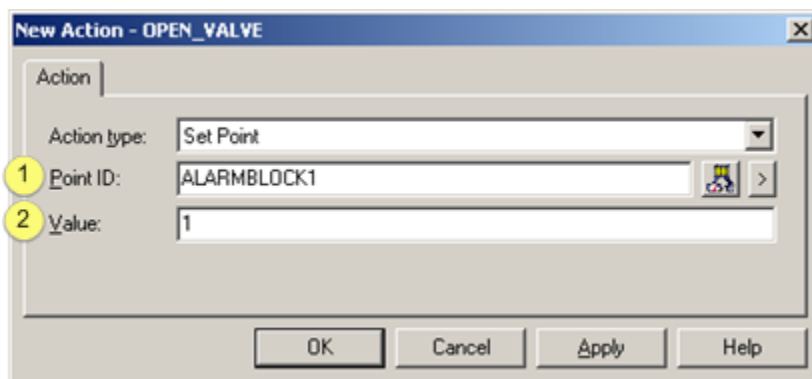
In the previous table:

- Scope of variable - Denotes the visibility of the variable to other script files, such as modules and AppDomains.
- Lifetime of variable value - Denotes how long the value of a variable will last.
- Shared - Denotes if two or more event instances will share the value of the variable.
- Multi-threading issues - Denotes if multi-threading issues occur when multiple instances run at the same time.

Set Point Actions

A **Set Point** action sets the value of a point.

To create this Action, enter the following information in the New Action dialog box:



1	Point ID
---	----------

ID of a point that will perform the set point.

(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Point browser.
	Popup Menu button	Displays Popup menu to create a new point, browse for or edit an existing point.

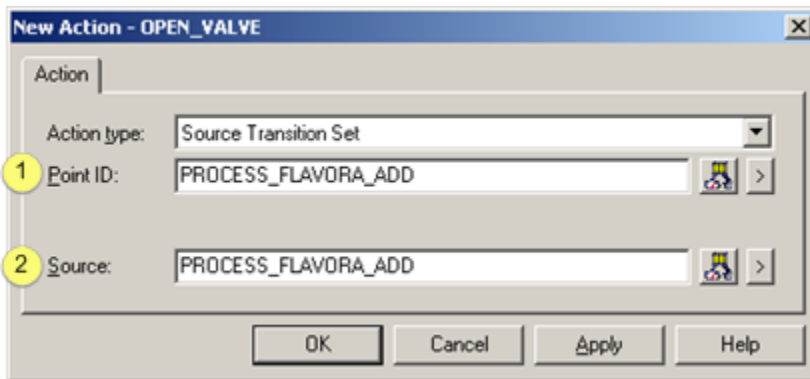
2	Value
---	-------

Value to set the point to.

Source Transition Set Actions

A Source Transition Set action sets the value of the point in the **Point ID** field to the value of the point in the **Source** field.

To create this Action, enter the following information in the New Action dialog box:



1	Point ID
---	----------

ID of a point that will perform the set point.

(Optional) Click either of the following to select the alarm ID.

	Browse button	Opens the Point browser.
--	---------------	--------------------------

	Popup Menu button	Displays Popup menu to create a new point, browse for or edit an existing point.
--	-------------------	--

2 Source

Name of the Point ID the will provide the update value.



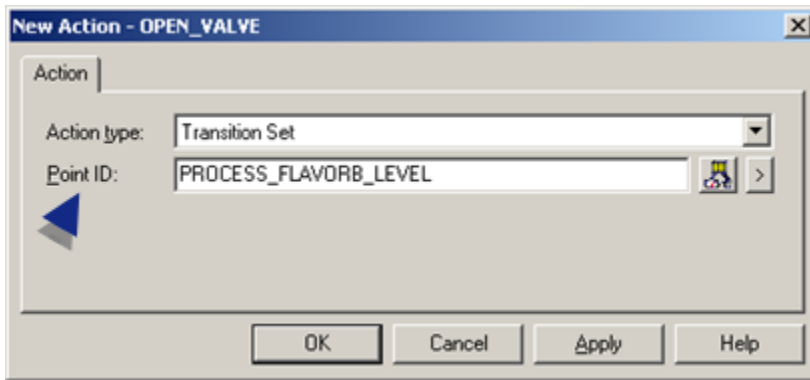
Note:

If the source point is the same as the point that triggered the event, the old value of the source point will be copied to the point ID. This lets you save a point value before it is updated. If you want to copy the new value of the point, use the [Transition Set \(on page 73\)](#) action.

Transition Set Actions

A **Transition Set** action sets the value of the point in the Point ID to the value of the point in the **Point ID** field of the Event associated with this Action.

To create this Action, enter the following information in the New Action dialog box



Point ID

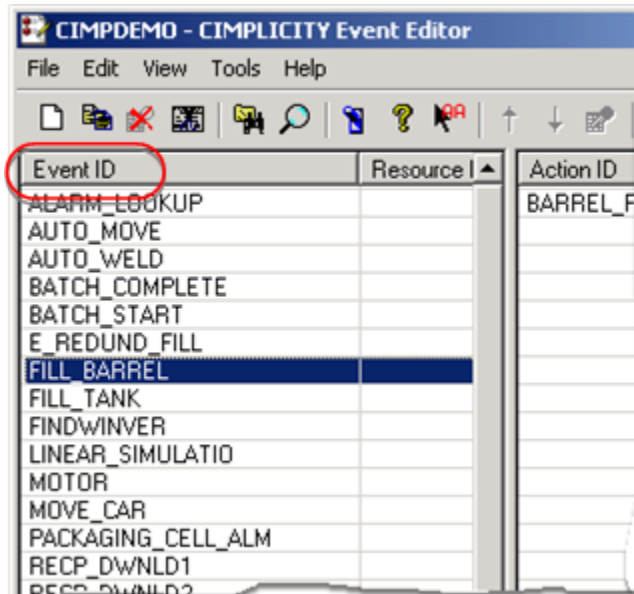
ID of a point that will be updated with the value of an associated event's point ID. Valid entries are either a device or global point ID

(Optional) Click either of the following to select the alarm ID.

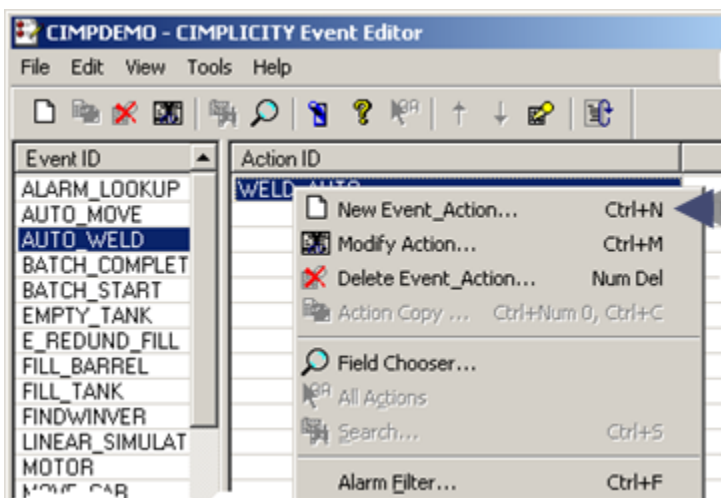
	Browse button	Opens the Point browser.
	Popup Menu button	Displays Popup menu to create a new point, browse for or edit an existing point.

Step 5. Associate Actions with an Event

1. Make sure you are displaying the Event Editor [By Event](#). ([on page 38](#))
2. Select an event in the Event list.



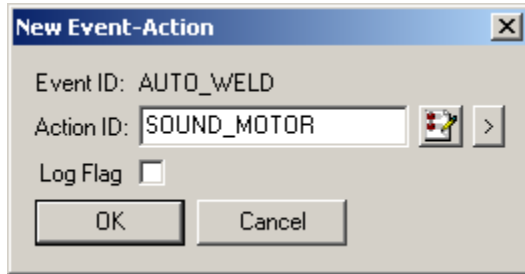
3. Click the mouse once in the Action list.
4. Do the following.
 - a. Right-click an event in the right pane.
 - b. Do one of the following.
 - Select New Event Action on the popup menu.



- Select [New Event-Action \(on page 36\)](#) on the File menu.
- Press Ctrl+C on the keyboard.

The New Event-Action dialog box opens.


5. Configure options are as follows.



Field	Description	
Event ID	Read-only	Event with which the action will be associated.
		Opens the Action browser.
		Displays popup menu to create a new action, browse for or edit an existing action.
Log Flag	Checked	Logs the event and action to the Database Logger Event Management log.
	Clear	Disables logging.



Tip:

You can also use the Toggle Logging button  on the Event Editor toolbar to enable or disable logging the selected event and action.

Step 6. Work with Existing Events and Actions

Step 6. Work with Existing Events and Actions

<p>Option 6.1 (on page 76)</p>	<p>Modify an event.</p>
--	-------------------------

Option 6.2 (on page 77)	Modify an action.
Option 6.3 (on page 79)	Copy an event.
Option 6.4 (on page 81)	Copy an action.
Option 6.5 (on page 83)	Filter alarms and events.
Option 6.6 (on page 89)	Select event display fields.
Option 6.7 (on page 91)	Select action display fields.
Option 6.8 (on page 93)	Search for an event.
Option 6.9 (on page 95)	Search for an Action


Option 6.1. Modify an Event

1. Select an event in the Event Viewer.

Note: The action can be selected in either Event or Action view.

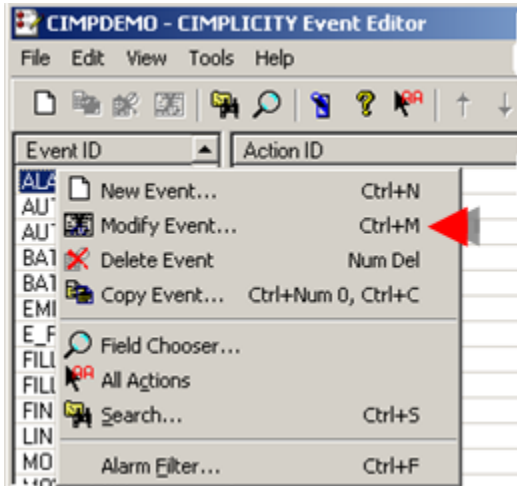
2. Do one of the following.

Method 1

Click the Modify  button on the Event Editor toolbar.

Method 2

- a. Right-click the selected event.
- b. Select Modify Event on the Popup menu.



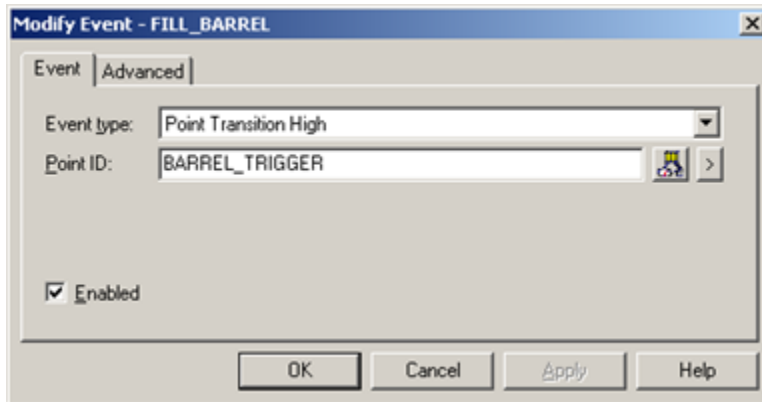
Method 3

Select Modify Event on the Event Editor [Edit menu \(on page 37\)](#).

Method 4

Press Ctrl+M on the keyboard.

A Modify Event dialog box opens with the configuration for the selected event.



3. Change any of the fields as you did when you [created \(on page 41\)](#) the event.


Option 6.2. Modify an Action

1. Select an action in the Event Viewer.

Note: The action can be selected in either Event or Action view.

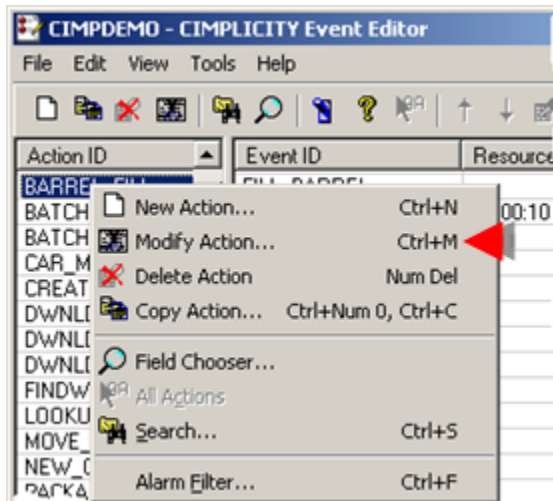
2. Do one of the following.

Method 1

Click the Modify  button on the Event Editor toolbar.

Method 2

- a. Right-click the selected action.
- b. Select Modify Action on the popup menu.



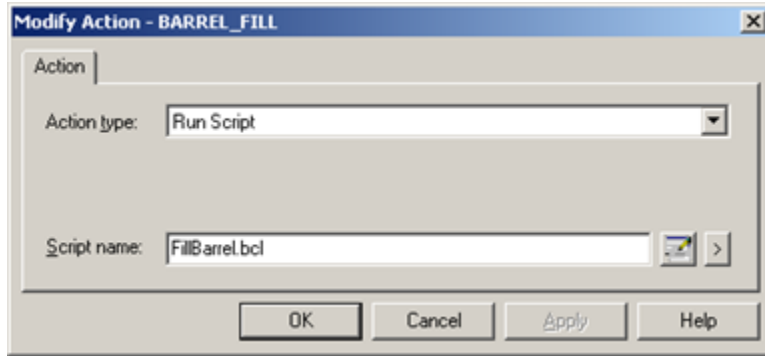
Method 3

Select Modify Acton on the Event Editor [Edit menu \(on page 37\)](#).

Method 4

Press Ctrl+M on the keyboard.

A Modify Action dialog box opens with the configuration for the selected event.



3. Change any of the fields as you did when you [created \(on page 60\)](#) the action.

Option 6.3. Copy an Event

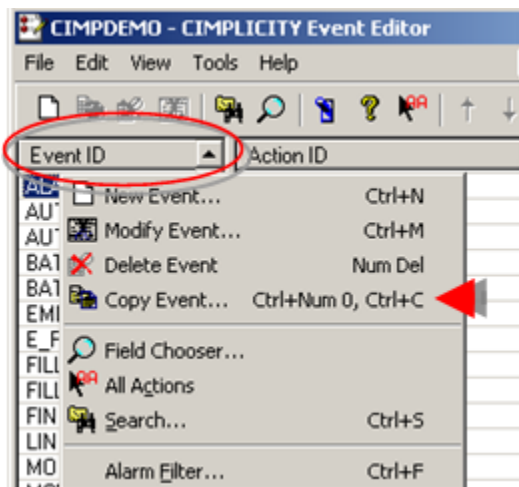
1. Click View on the CIMPLICITY Event Editor menu bar.
2. Select [By Event \(on page 38\)](#).
3. Select the Event in the Event list.
4. Do one of the following.

Method 1

Click the Copy button  on the Event Editor toolbar.

Method 2

- a. Right-click the selected event.
- b. Select Copy Event on the popup menu.



Method 3

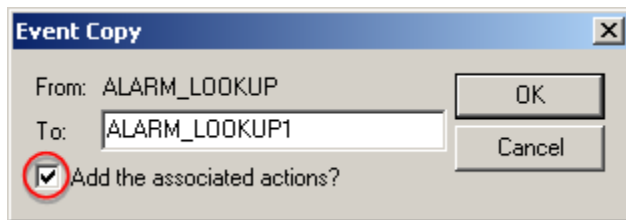
Select Copy Event on the Event Editor [Edit menu \(on page 37\)](#) .

Method 4

Press Ctrl+C on the keyboard

The Event Copy dialog box opens.

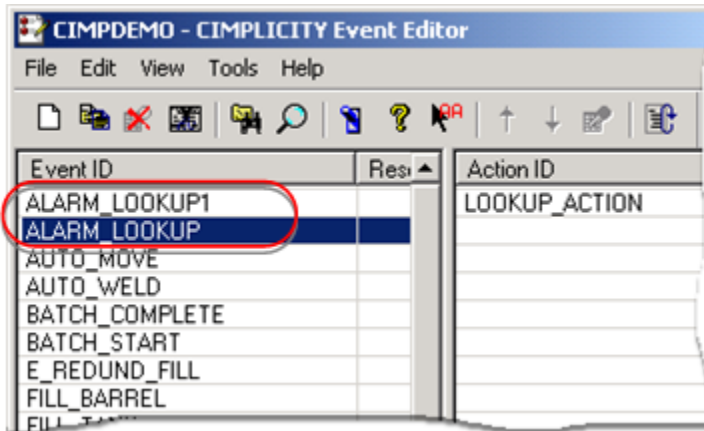
5. Make selections are as follows.



Selection	Description	
From	(Read only) Selected event.	
To	Name of the event to which the selected event's configuration will be copied.	
Add the associated actions?	Checked	Copies all actions associated with the source event to the target event.
	Unchecked	Copies only the event configuration; does not copy associated actions.

6. Click OK

The dialog box closes and the new Event appears on the Event list.



Option 6.4. Copy an Action

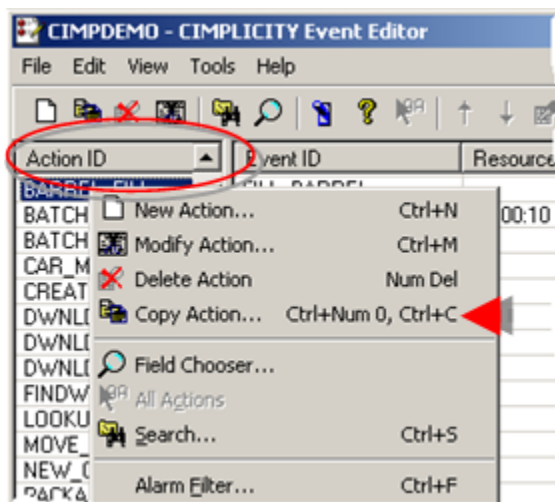
1. Click View on the Event Editor menu bar.
2. Select *by Action (on page 38)*.
3. Do one of the following.

Method 1

Click the Copy button  on the Event Editor toolbar.

Method 2

- a. Right-click the selected action.
- b. Select Copy action on the popup menu.



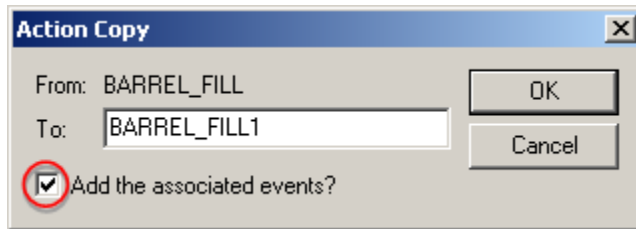
Method 3

Select Action Copy on the Event Editor Edit menu.

Method 4

Press Ctrl+C on the keyboard.

The Action Copy dialog box opens.

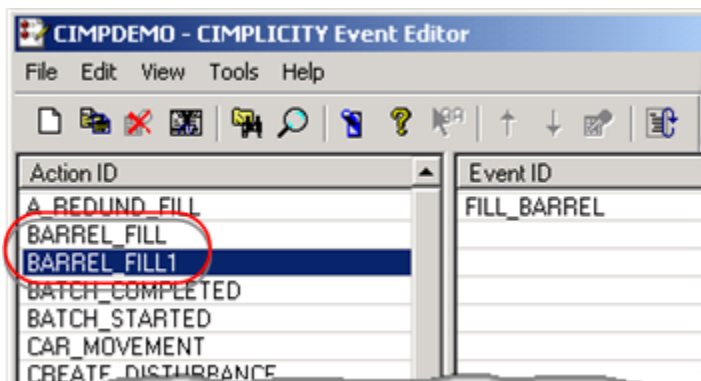


4. Make selections are as follows.

Selection	Description	
From	(Read only) Selected action.	
To	Name of the action to which the selected action's configuration will be copied.	
Add the associated events?	Checked	Copies all events associated with the source action to the target event.
	Unchecked	Copies only the action configuration; does not copy associated events..

5. Click OK.

The dialog box closes and the new Action appears on the Action list.



Option 6.5. Filter Alarms and Events

The Alarm Setup dialog box lets you filter the alarms to which the Event Manager will respond.

You can filter by:

- Resource ID
- Alarm Class ID.

You can also have the Event Manager respond to either or both;

- Alarm Log data
- Event Log data



Important:

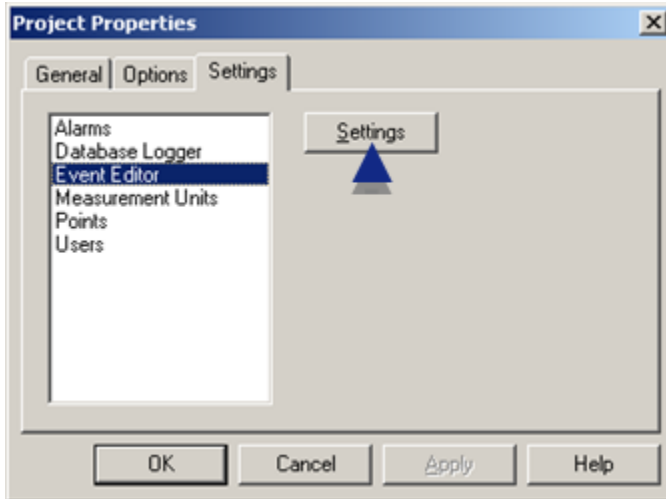
You must enter information in the [Setup \(on page 84\)](#) dialog box in order to receive alarm and/or event data.

Open the Alarm Setup Dialog box

Do one of the following to open the Alarm Setup dialog box. The Setup dialog box opens when you use any of these methods.

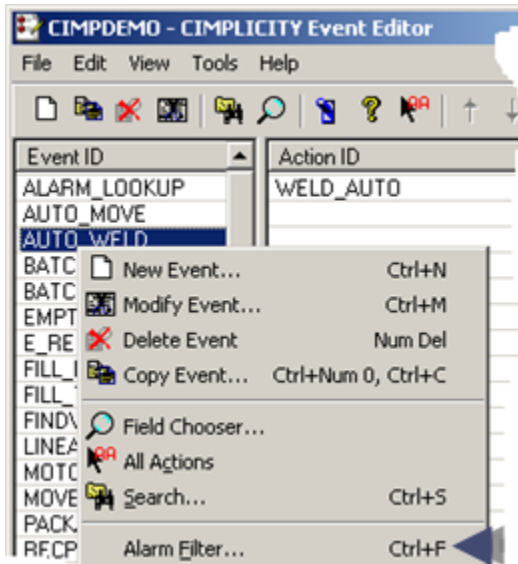
Method 1:

- Open the Project Properties dialog box.
- Select the Settings tab.
- Select Event Editor.
- Click Settings.



Method 2

- Right-click an Event ID in the CIMPLICITY Event Editor left pane
- Select Alarm Filter from the pop-up menu.

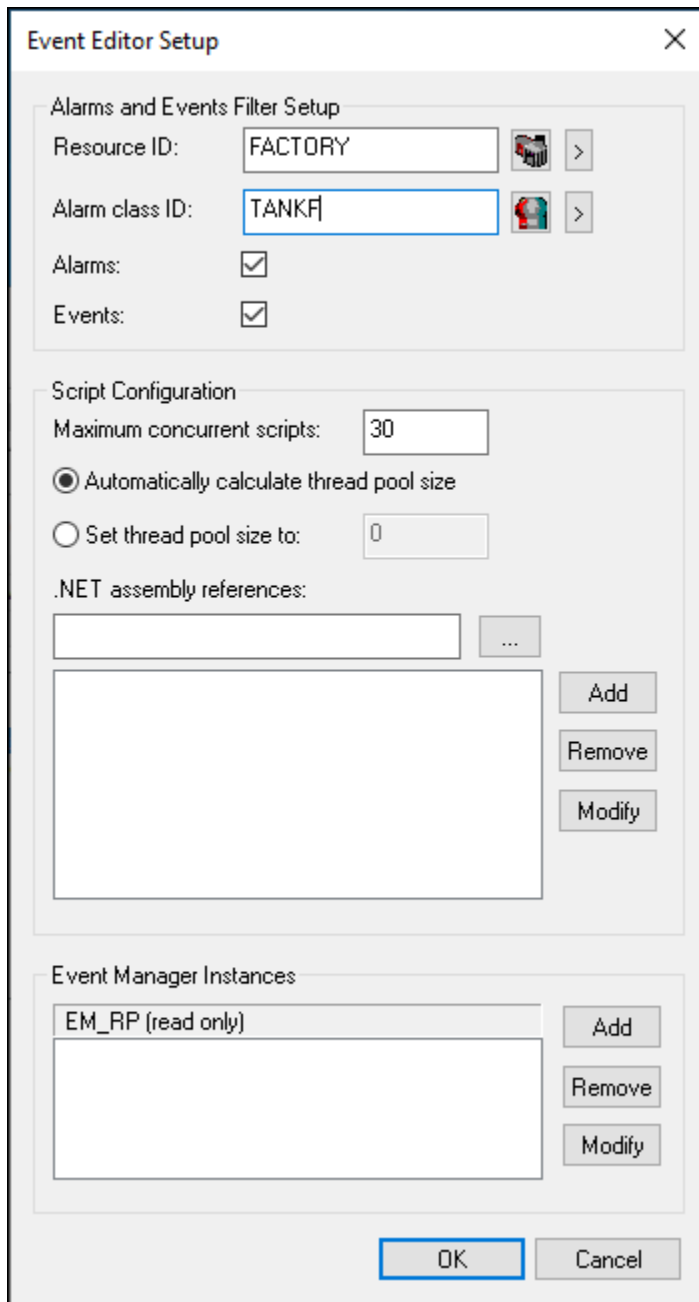



Method 3




- Select Alarm Filter on the CIMPLICITY [Event Editor Edit \(on page 37\)](#) menu.
- Method 4
- Press **Ctrl+F** in the CIMPLICITY Event Editor.

Setup Options

Setup options are as follows.



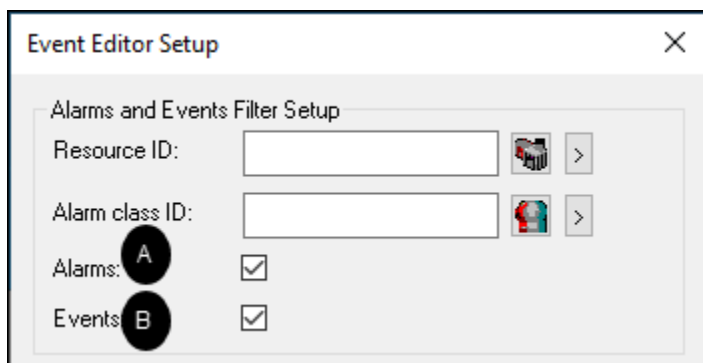
Option	Description
Resource ID	Resource ID for which the Event Manager can receive information.
	Opens the Resource browser.

Option	Description	
		Displays window to create a new resource, browse for or edit an existing resource.
Alarm Class ID	Alarm Class for which the Event Manager can receive information.	
		Opens the Alarm Class browser.
		Displays window to create a new alarm, browse for or edit an existing resource.
Alarms	Selected	The Event Manager will receive Alarm Log data.
	Cleared	The Event Manager will not receive any Alarm Log data.
Events	Selected	The Event Manager will receive Event Log data.
	Cleared	The Event Manager will not receive any Event Log data.
Maximum Concurrent Scripts	<p>Specifies the maximum number of scripts that can execute concurrently within the Event Manager. When this limit is exceeded:</p> <ul style="list-style-type: none"> • An \$EM_MAX_SCRIPTS alarm will be generated. • A Too many executing threads, action ignored message will appear in the status log. 	
Maximum concurrent scripts	<p>Specifies the maximum number of scripts that can execute concurrently within the Event Manager.</p> <p>When this limit is exceeded:</p> <ul style="list-style-type: none"> • An \$EM_MAX_SCRIPTS alarm will be generated. • A Too many executing threads, action ignored message will appear in the status log. 	
Automatically calculate thread pool size	Sets the thread pool size to twice the number of logical processors in the system.	

Option	Description
Set thread pool size to	Sets the thread pool size to a value between 0 and 200 that you enter. If you enter a value that is greater than twice the number of logical processors in the system, a warning is displayed.
.Net Assembly References (on page 267)	Additional .NET assembly references for C# and VB .NET; additional references can be added or removed
Event Manager Instances (on page 97)	You can configure multiple Event Manager instances in a project. This allows logical separation of critical/key events into their own instances which can prevent unwanted interactions.

Example 1

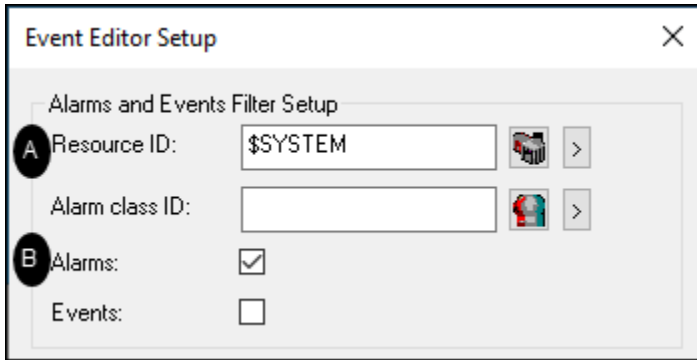
Enter and select the following to make the Event Manager receive all alarms and events.



Letter	Option	Action
	Resource ID	Leave blank.
	Alarm class ID	Leave blank.
A	Alarms	Check
B	Events	Check

Example 2

Enter and select the following to make the Event Manager receive only Event data for system resources.

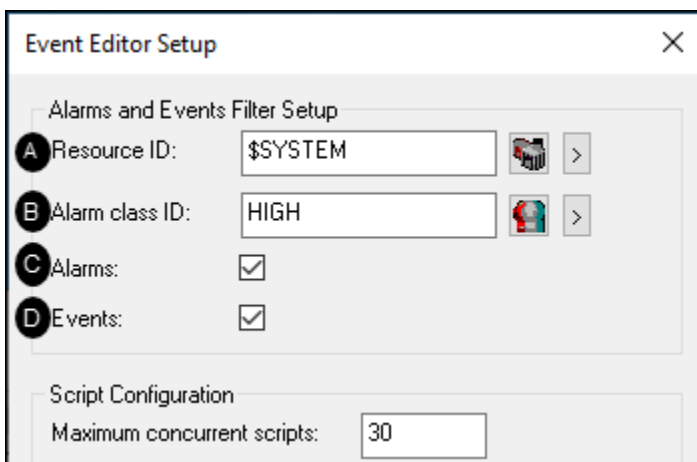


Letter	Option	Action
A	Resource ID	Enter \$SYSTEM
	Alarm class ID	Leave blank.
B	Alarms	Clear
	Events	Check

Example 3

Enter the following to make the Event Manager:

- Receive event data from the \$SYSTEM resource.
- Receive HIGH class alarm data only from \$SYSTEM resource.



Letter	Option	Action
A	Resource ID	Enter \$SYSTEM.
B	Alarm class ID	Enter HIGH.

Letter	Option	Action
C	Alarms	Check
D	Events	Check

Option 6.6. Select Event Display Fields

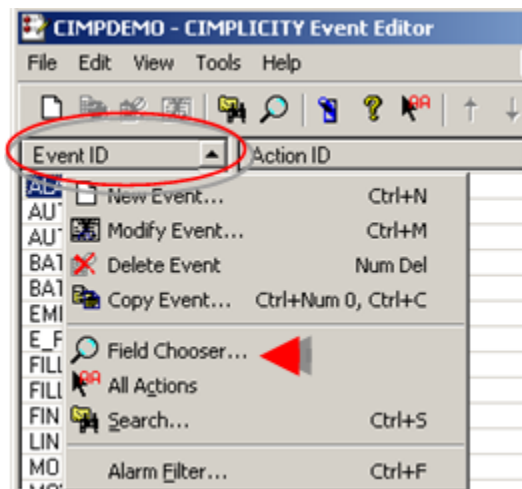
1. Click View on the CIMPLICITY Event Editor menu bar.
2. Select [By Event \(on page 38\)](#).
3. Do one of the following.

Method 1

Click the Field Chooser button  on the Event Editor toolbar.

Method 2

- a. Click the right mouse button in the Event Editor left pane.
- b. Select Field Chooser on the popup menu.

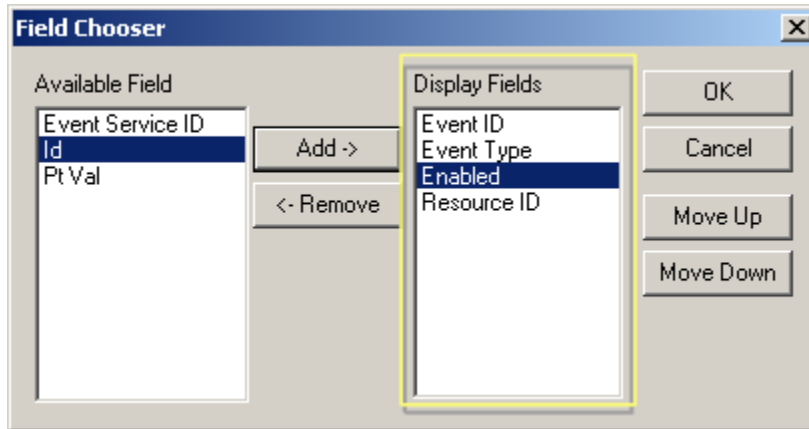


Method 3

Select Field Chooser on the Event Editor [View menu \(on page 38\)](#) .

Result: The Field Chooser dialog box for the Event field columns opens.

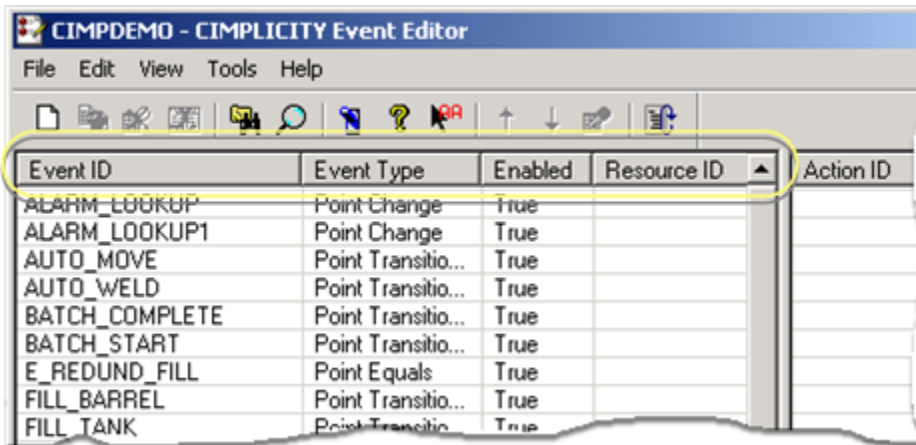
Field Chooser features are as follows.



Feature	Description	
Lists	Available Field	Event fields that are not currently being displayed.
	Display Fields	Event fields that are currently being displayed. The fields display in columns. Columns go from left to right, starting at the top of the list and moving down.
Fields	Selections correspond to the selections in the Event dialog boxes (on page 42) .	
Buttons	Add	Add selected available fields to the Display Fields list
	Remove	Stops displaying selected fields by sending them back to the Available Field list.
	OK	Saves the selection and closes the Field Chooser
	Cancel	Cancel the latest selections.
	Move Up	Each click moves a selected field one column to the left. Note: Event ID is always the farthest left.
	Move Down	Each click moves a selected field one column to the right.

4. Click OK when you have finished making your selections.

The Event Editor left pane displays your selections.



Option 6.7. Select Action Display Fields

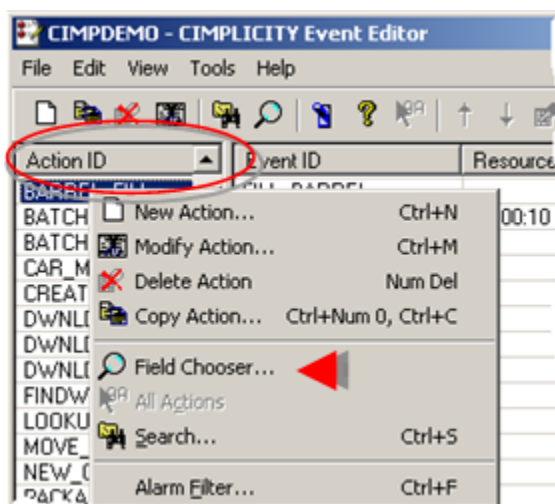
1. Click View on the Event Editor menu bar.
2. Select *by Action (on page 38)*.
3. Do one of the following.

Method 1

Click the Field Chooser button  on the Event Editor toolbar.

Method 2

- a. Click the right mouse button in the Event Editor left pane.
- b. Select Field Chooser on the popup menu.

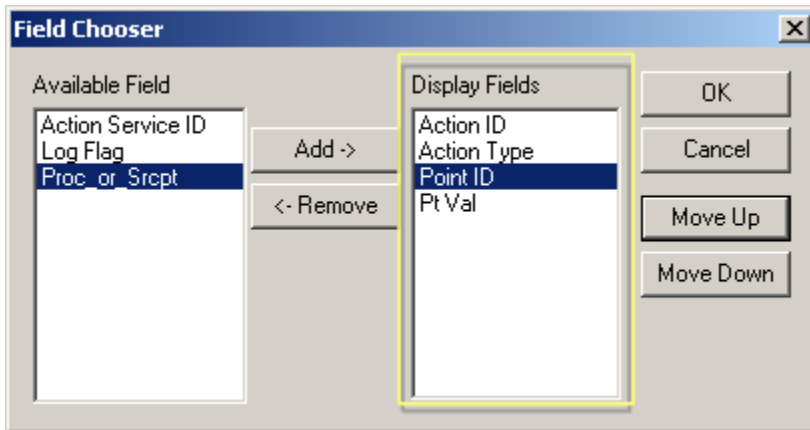


Method 3

Select Field Chooser on the Event Editor [View menu \(on page 38\)](#).

Result: The Field Chooser dialog box for the Action field columns opens.

Field Chooser features are as follows.

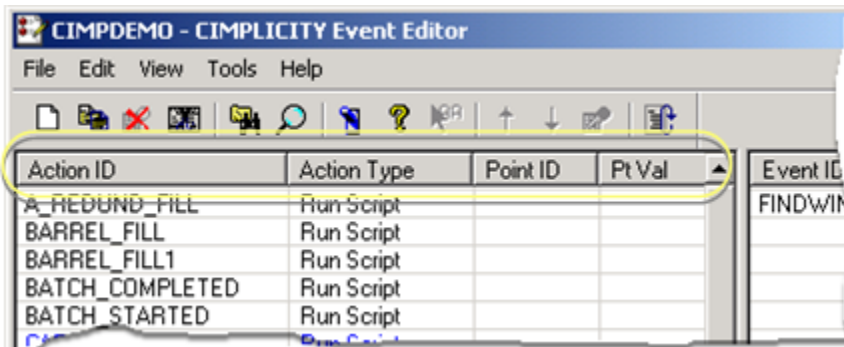


Feature	Description	
Lists	Available Field	Action fields that are not currently being displayed.
	Display Fields	Action fields that are currently being displayed. The fields display in columns. Columns go from left to right, starting at the top of the list and moving down.
Fields	Selections correspond to the selections in the Action dialog boxes (on page 60) .	
Buttons	Add	Add selected available fields to the Display Fields list
	Remove	Stops displaying selected fields by sending them back to the Available Field list.
	OK	Saves the selection and closes the Field Chooser
	Cancel	Cancel the latest selections.

Move Up	Each click moves a selected field one column to the left. Note: Action ID is always the farthest left.
Move Down	Each click moves a selected field one column to the right.

- Click OK when you have finished making your selections.

The Event Editor left pane displays your selections.



Option 6.8. Search for an Event

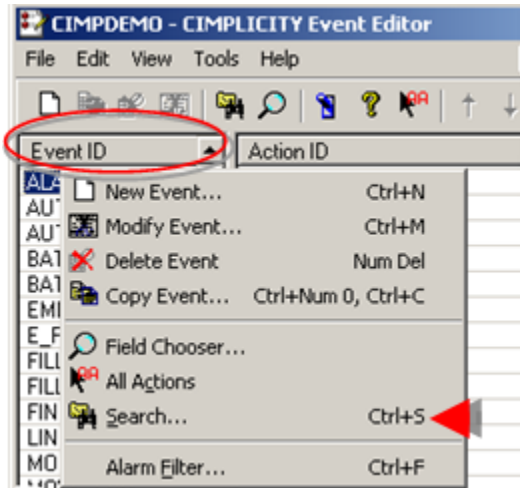
- Click View on the CIMPLICITY Event Editor menu bar.
- Select [By Event \(on page 38\)](#).
- Do one of the following.

Method 1

Click the Search button  on the Event Editor toolbar.

Method 2

- Right-click the Event Editor left pane.
- Select Search on the popup menu.



Method 3

Select Search on the Event Editor [View menu \(on page 38\)](#).

Method 4

Press Ctrl+S on the keyboard.

A Event Search dialog box opens.

Search criteria are as follows.

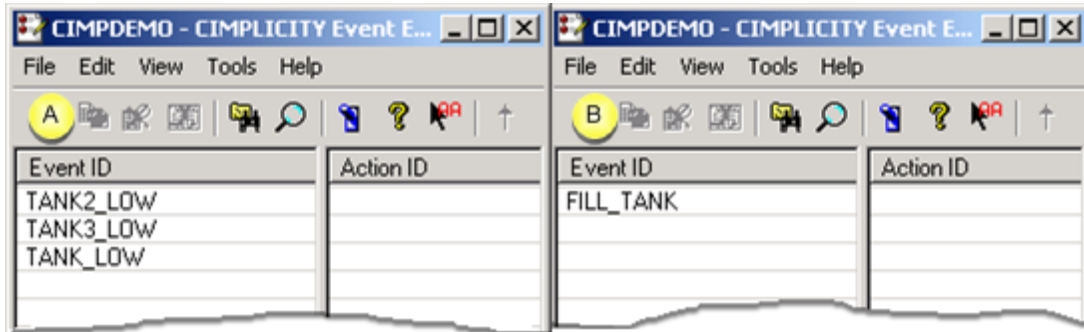


	Field	Description
A	Event ID	ID or partial ID with a * wild card of the event or events you want to find.

B	Id	ID of a point used in an event definition.
---	----	--

4. Click OK.

Events that fill your criteria display in the Event Editor.



Tip:

Leave the fields blank in the Event search dialog box to re-display all of the events in the Event Editor after you have searched for selected events.

Option 6.9. Search for an Action

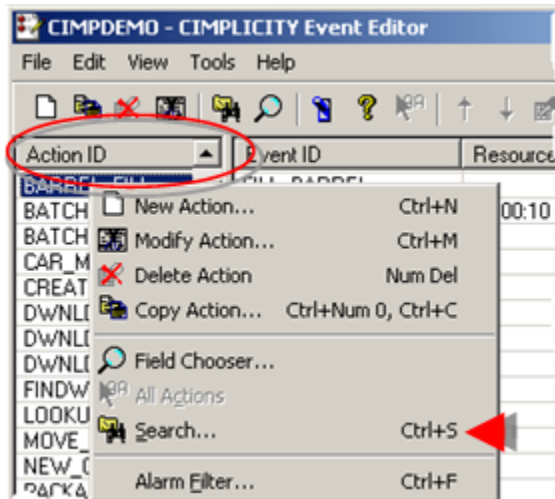
1. Click View on the CIMPLICITY Event Editor menu bar.
2. Select *By Action (on page 38)*.
3. Do one of the following.

Method 1

Click the Search button  on the Event Editor toolbar.

Method 2

- a. Right-click the Event Editor left pane.
- b. Select Search on the popup menu.



Method 3

Select Search on the Event Editor [View menu \(on page 38\)](#).

Method 4

Press Ctrl+S on the keyboard.

An Action Search dialog box opens.

Search criteria are as follows.

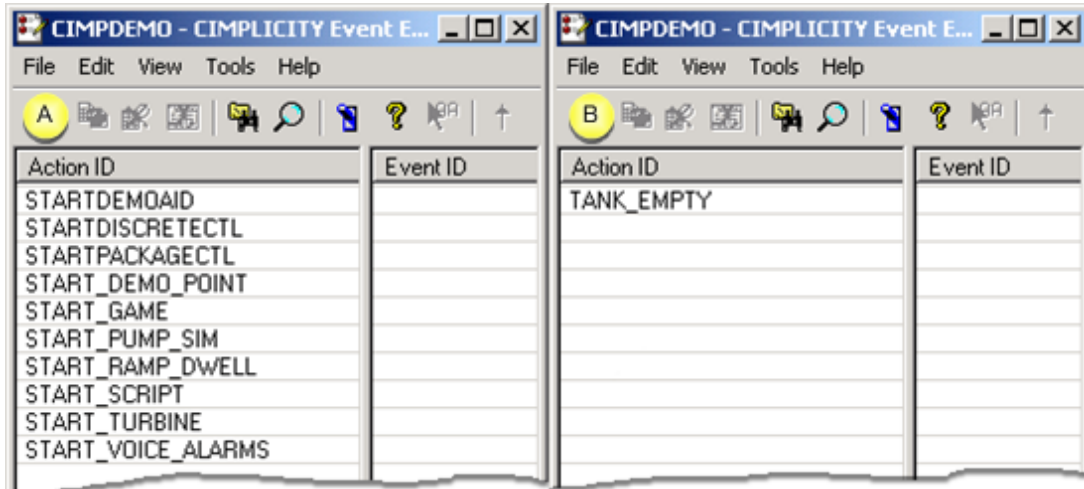


	Field	Description
A	Action ID	ID or partial ID with a * wild card of the action or actions you want to find.

B	Point Id	ID of a point used in an action definition.
---	----------	---

4. Click OK.

Actions that fill your criteria display in the Event Editor.



Tip:

Leave the fields blank in the Event search dialog box to re-display all of the actions in the Event Editor after you have searched for selected actions.

Configure Multiple Event Manager Instances in a Project

About Multiple Event Manager Resident Process (EMRP)

Multiple Event Manager instances in a project enables logical separation of critical events into their own instances and prevent unintended interactions.

Having EMRP has the following advantages:

- Better scaling for Basic Script-based scripts can be achieved through multiple Event Manager instances. Each instance receives its own string-space pool, overcoming string-space limitations.
- Isolation of critical events allows you to run your own processes and have dedicated resources.
- Multiple Event Manager instances are not impacted by the activities of less important scripts.
- Allow grouping of scripts that share common resources like database connections, criticality, and more.

Important Points to Consider before configuring Event Manager Instances

- You can create a maximum of 63 EMRP instances in each project in addition to the default EMRP instance. For example, **EM_RP** (default), **EM_RP1**, **EM_RP2**, ..., **EM_RP63**.
- The instance can be of any name, however, it is recommended to use a simple, and meaningful instance name.



Note:

You cannot use special characters such as | or \$, and the instance name cannot begin with @.

- The instance name can be of a maximum of 32 characters.
- The Event Manager instance that you create is applicable to alarms and events.
- You cannot add a new EMRP instance or change the service ID dynamically for alarms and events.
- You cannot delete the default Event Manager instance, that is, EM_RP, and the instance that is currently associated to an event.
- Adding multiple EMRP instances will have an impact on a project's startup time.

What to do next:

[Add a new EMRP Instance \(on page 98\)](#).

Add a New EMRP Instance

This topic describes how to add a new EMRP instance.

Before you add an EMRP instance, know the following:

- You can add a maximum of 63 EMRP instances in each project in addition to the default EMRP instance. For example, EM_RP (default), EM_RP1, EM_RP2, ..., EM_RP63.
- The instance can be of any name, however, it is recommended to use a simple, and meaningful instance name.

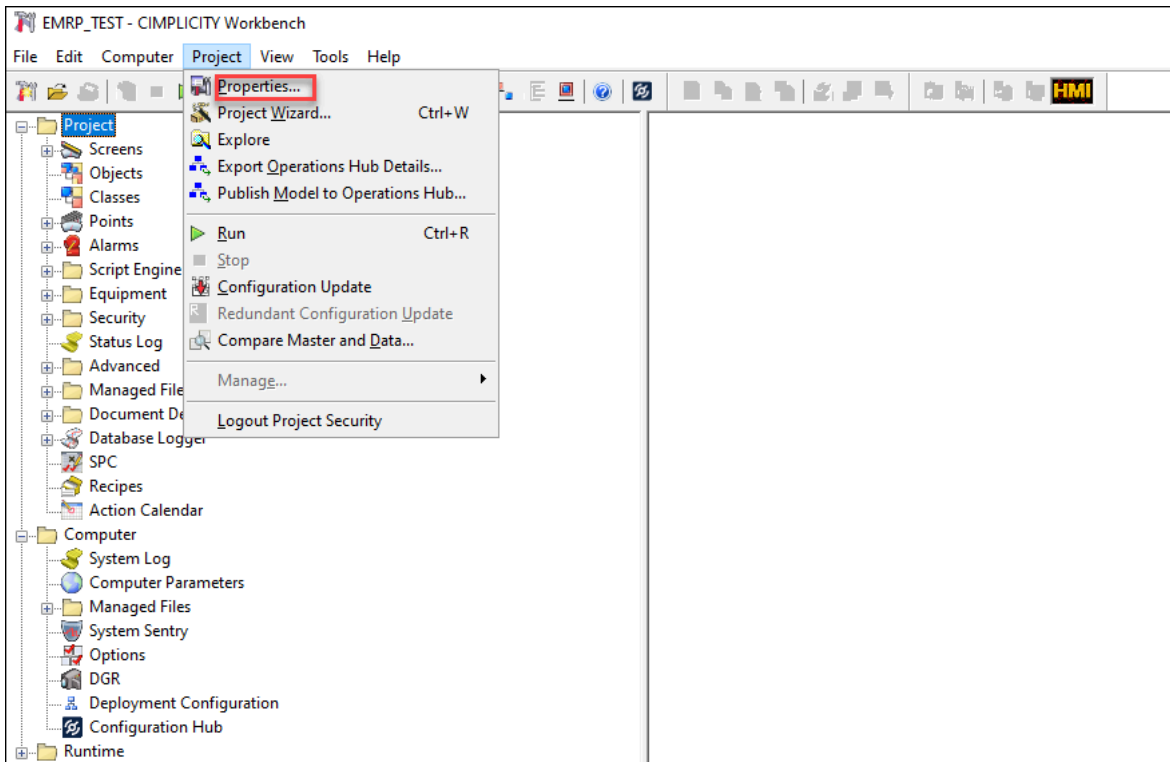


Note:

You cannot use special characters such as | or \$, and the instance name cannot begin with @.

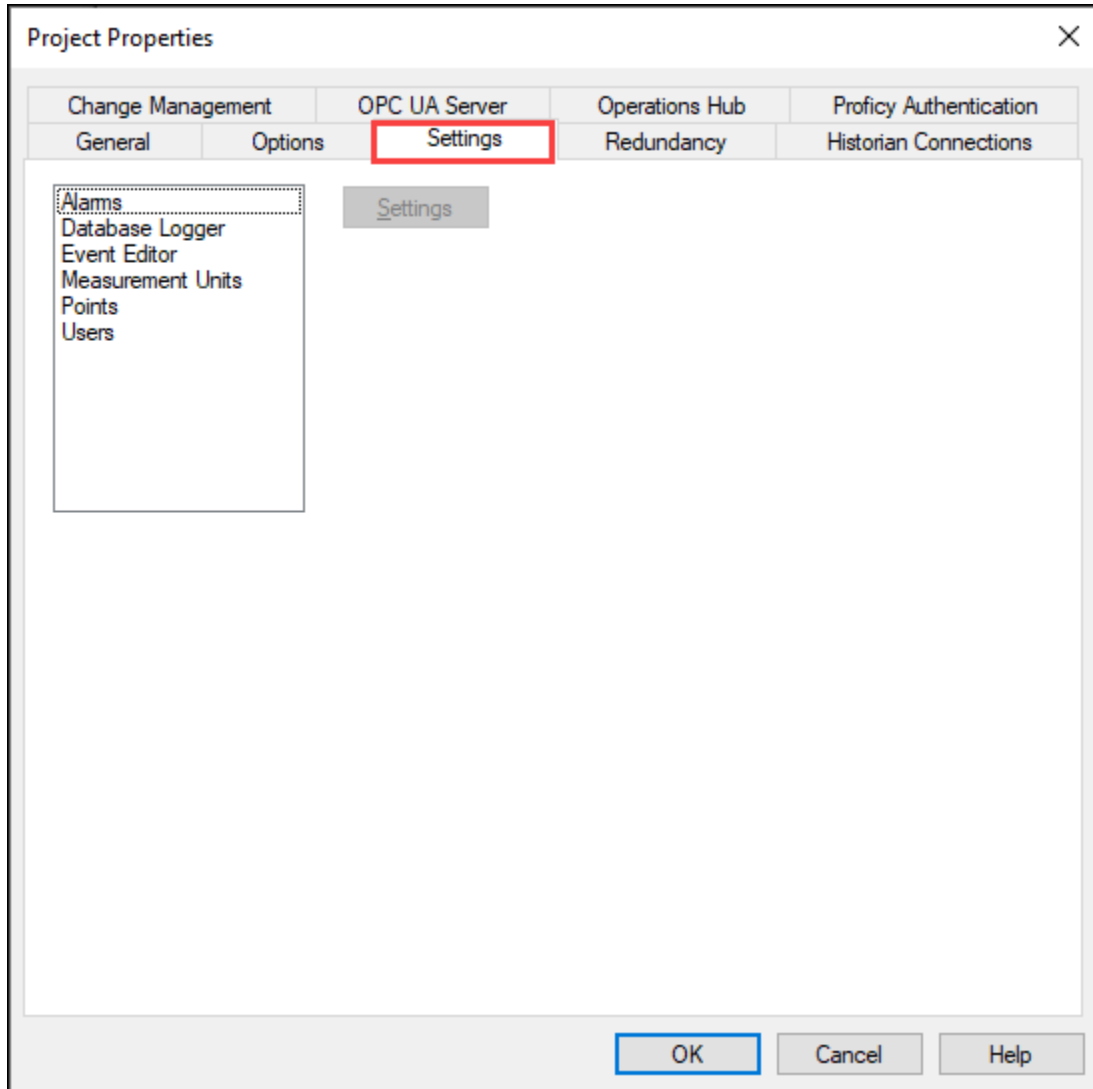
- The instance name can be of a maximum of 32 characters.
- Using the Dynamic Update button in the Event Editor does not support updating the Event Manager instance dynamically.

1. Open the CIMPLICITY project in the CIMPLICITY Workbench.
The CIMPLICITY Workbench opens, loading the project.
2. In the menu, select **Project**, and then select **Properties**.

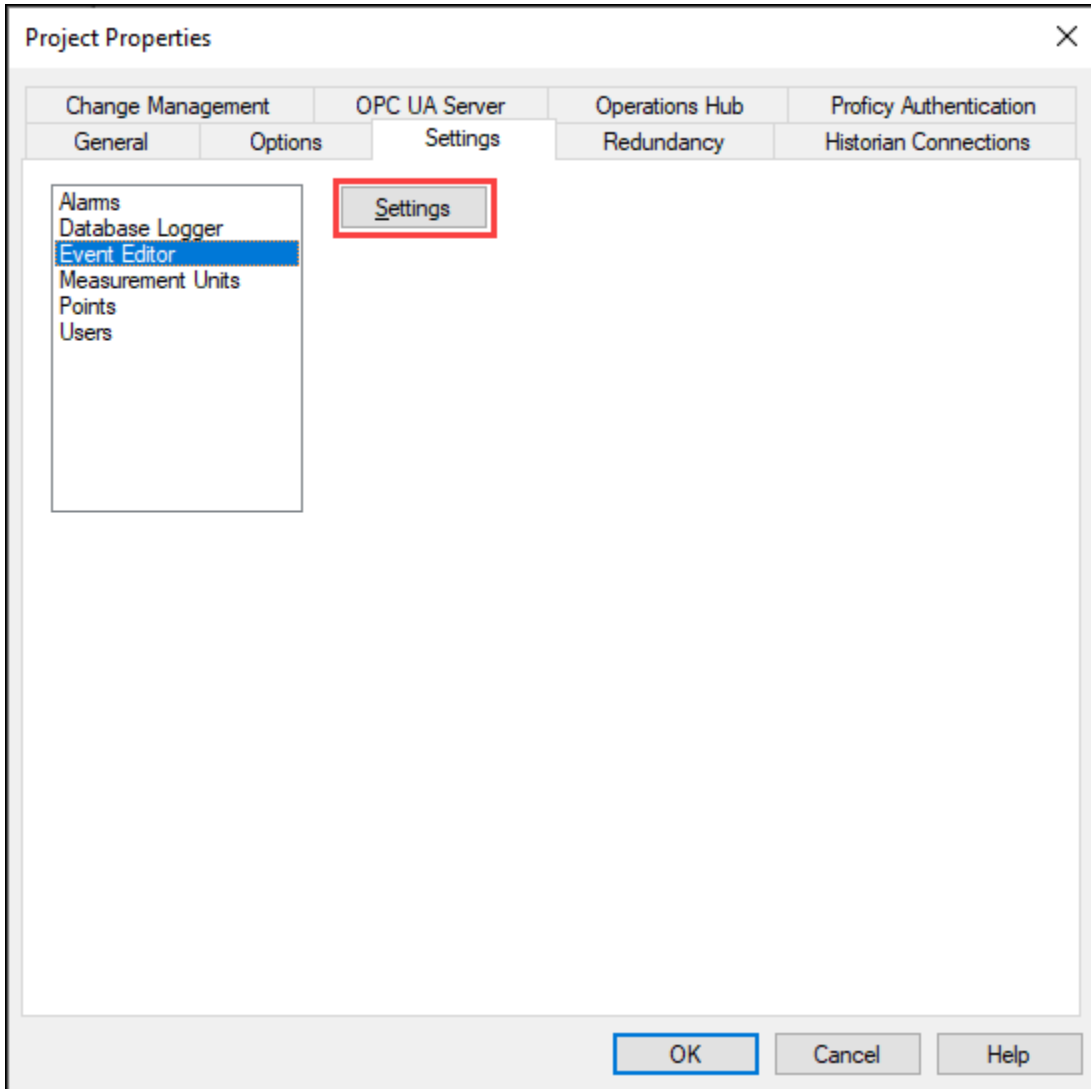


The **Project Properties** window opens.

3. Select the **Settings** tab.




4. Select **Event Editor**, and then select **Settings**.




The **Event Editor Setup** window opens.

Event Editor Setup

Alarms and Events Filter Setup

Resource ID:  >

Alarm class ID:  >

Alarms:

Events:

Script Configuration

Maximum concurrent scripts:

Automatically calculate thread pool size

Set thread pool size to:

.NET assembly references:

...

Event Manager Instances

5. In the **Event Manager Instances**, select **Add**.

The text area to add a new instance name is enabled.

6. Type a name for the EMRP instance. For example, EM_RP1.
7. On your keyboard, press **Enter**.

The new EMRP instance is added.

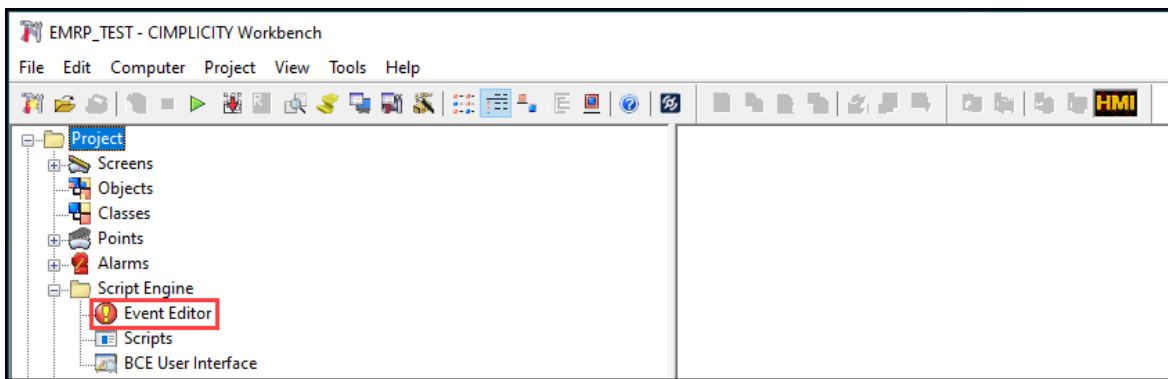
8. If needed, add more EMRP instances as needed.
 9. If you have added the EMRP instances as needed, select **OK** to exit the **Event Editor Setup** window.
- This will register a new Event Manager process that will start when you start the project.

[Associate the Event Manager instance to an event \(on page 103\).](#)

Associate Event Manager Instance to an Event

This topic describes how to associate an Event Manager instance to an event. For example, this topic describes creating a new event and associating an Event Manager instance to it. You can also associate the Event Manager instance to an existing event by opening the event's properties window in the Event Editor.

1. In the CIMPLICITY Workbench, under **Project**, expand **Script Engine**, and then double-click **Event Editor**.

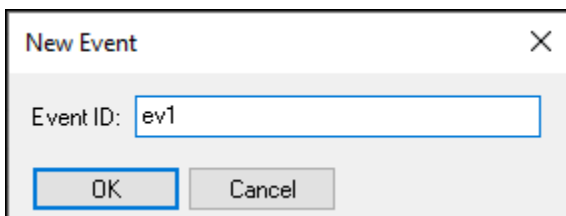


The **CIMPLICITY Event Editor** window opens.

2. In the menu, select **File** and then select **New Event**.

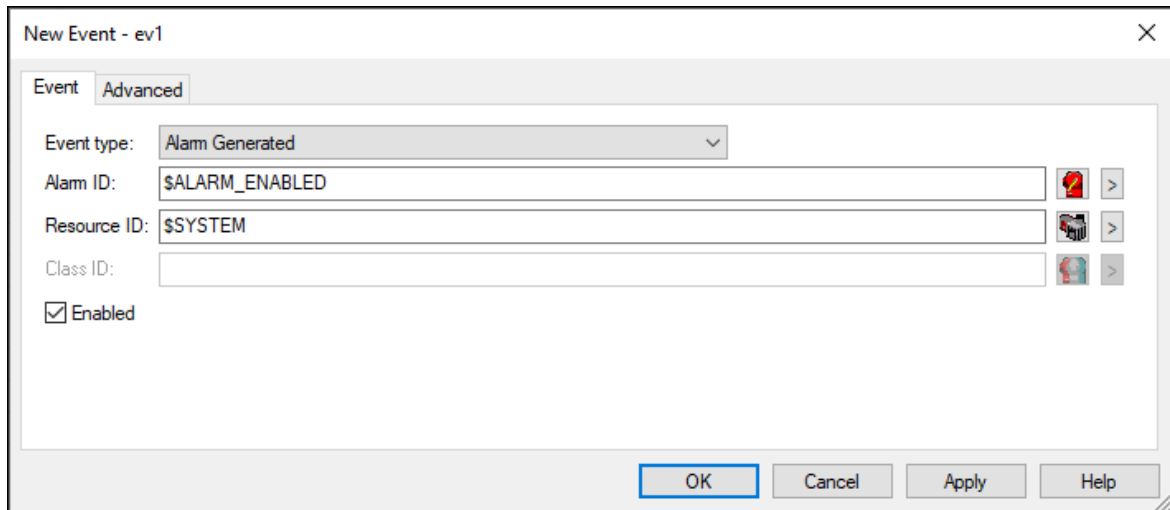
Alternatively, you can either select the new file icon in the tool bar, or you can use the Ctrl+N keyboard shortcut.

The **New Event** window opens.

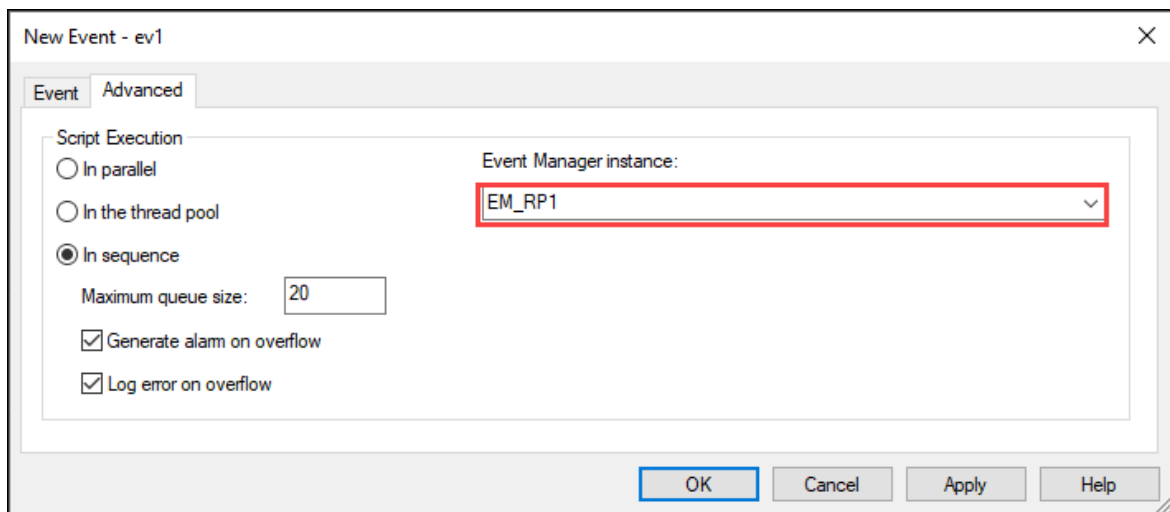


3. Type an event ID. For example, ev1.
4. Select **OK**.

The **New Event** properties window opens.



5. On the **Event** tab, configure the event as per the event type and other fields as needed.
6. Select the **Advanced** tab.
7. From the **Event Manager instance** drop-down, select an Event Manager instance that you added.



8. Select **Apply**, and then select **OK**.

The new Event Manager instance will be associated to the event.

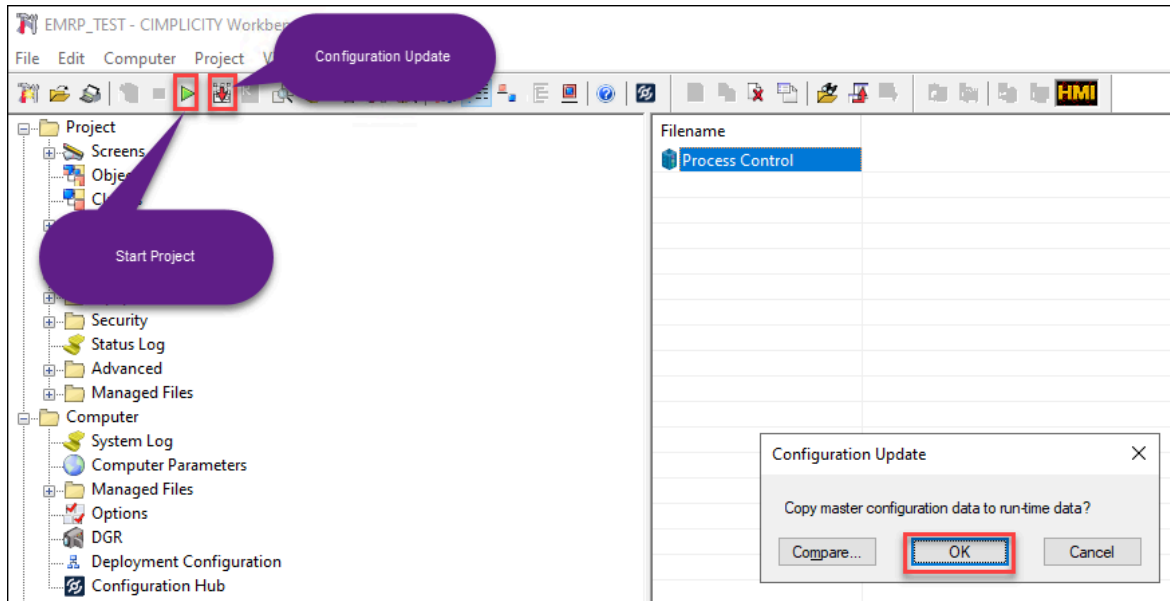
Did you know that you could also associate an Event Manager instance to a class event. For more information, refer to [Class Event Definition](#) (on page [104](#)).

[Check if the new event manager instance is running](#) (on page [104](#)).

Check if the New Event Manager Instance is Running

This topic describes how to check the if the created Event Manager instance is running.

1. In the CIMPLICITY workbench, do a Configuration Update and start the project.

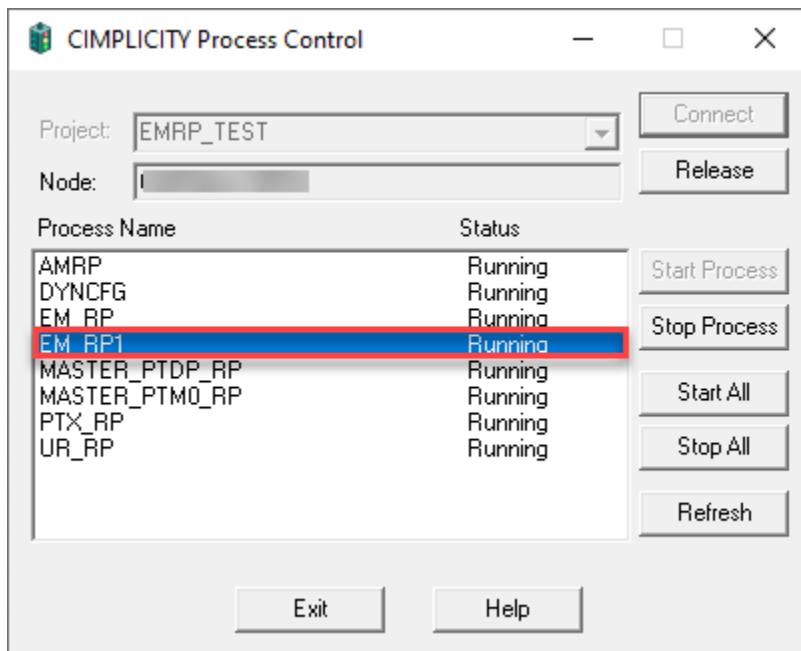


2. After the project is started, under **Runtime**, select **Process Control Panel**.

The **CIMPLICITY Process Control** window opens.

3. Select the project and then select **Connect**.

The connection to the project establishes, listing all the processes running in the project.



4. Confirm the created Event Manager instance is listed and running, in this example, EM_RP1.

If needed, you can also [test the events in Basic Control Engine UI \(on page 106\)](#).

Test the Newly Configured Event in Basic Control Engine User Interface (BCEUI)

This topic describes how to test if the newly configured event in BCEUI.



Note:

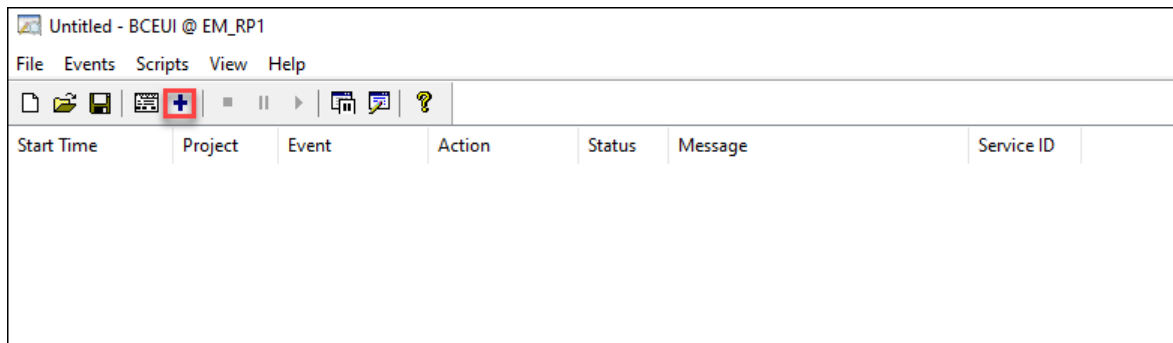
The BCEUI window can only show events from one Event Manager instance at a time. To see events from different Event Manager instances, you have to open a separate BCEUI window for each one.

1. In the CIMPLICITY Workbench, in the menu, select **Tools**, and then select **Command Prompt**.
The command prompt opens.
2. In the command prompt, run the following:

```
Bceui.exe /emrp=INSTANCE
```

For example, `Bceui.exe /emrp=EM_RP1`.

The BCEUI window opens.



3. To select an event, select **+**.
The **Select an Event** window opens.


File View

Project :

Event Service ID

Event ID

Id

Event ID	Resource ID
 ev1	\$SYSTEM

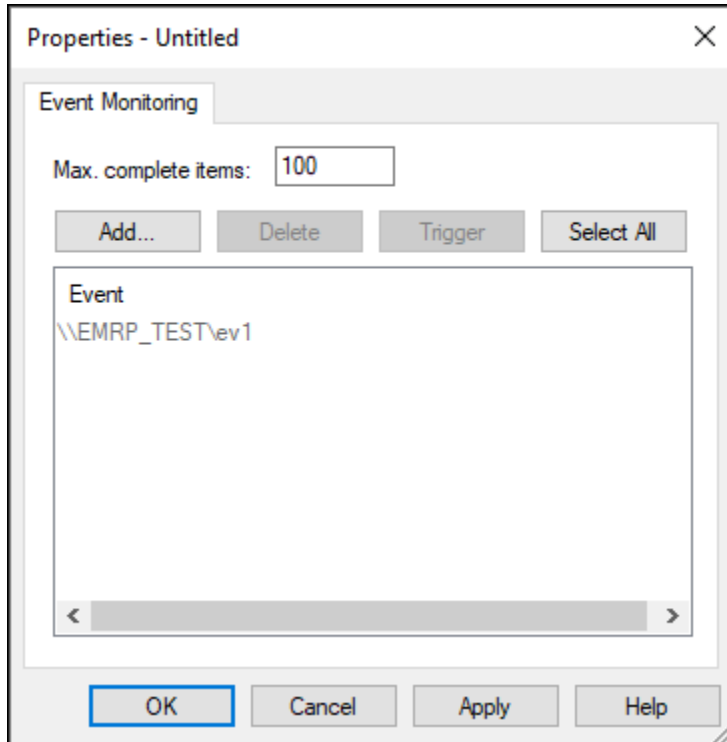
Records retrieved 1

Max. Record Limit

4. Select the project and the associated event ID.

5. Select **OK**.

The selected event ID is added to **Event Monitoring**.



6. Select the event, and select **Apply**, and then select **OK**.
The event will be monitored in the BCEUI.
7. Trigger the event to which the new EMRP instance is associated.
A line item with the event trigger and status should be displayed in the BCEUI.

Optimize Event Editor Performance

You can do the following to optimize the performance of the Event Manager:

- Make entries in the Global Parameters file to change the maximum number of scripts that can run simultaneously, or specify how long an idle thread should remain active.

Event Editor global parameters include:

CE_MAX_THREADS
CE_THREAD_TIMEOUT

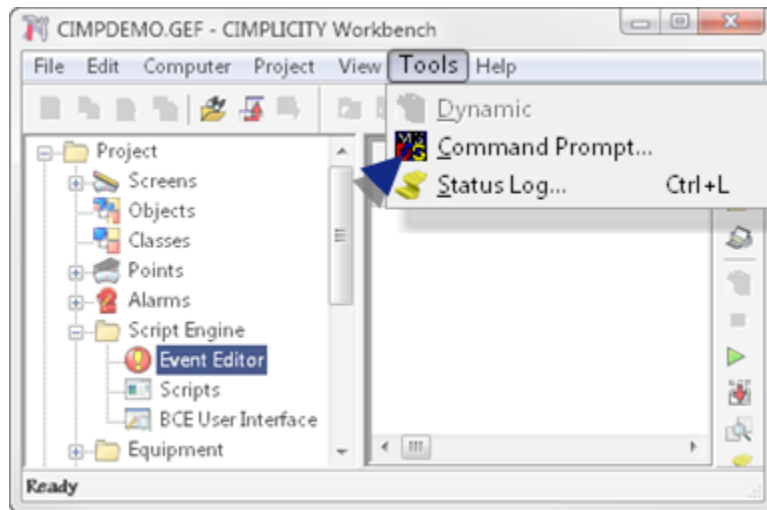
- Make entries in the Basic Control Engine points file to cache frequently used points.

Each time a script uses a point, it must retrieve the point's definition. You can use the `bce_points.cfg` file to pre-load point definitions into the Basic Control Engine for the Event Manager. This can provide a performance boost.

The `bce_points.cfg` file is an ASCII file that needs to be located in your project's Data directory.

To create the `bce_points.cfg` file:

1. Select the **Tools>Command Prompt** on the Workbench Tools menu.



2. Type `cd %SITE_ROOT%\data`.
3. Type `notepad bce_points.cfg`.

Notepad opens with a blank `bce_points.cfg` file loaded.

4. Enter all the point IDs that you want to cache, one per line in the file.
5. Exit Notepad and save the file.
6. Stop and restart your project to have the caching take effect.

Chapter 2. Script Editors

About Script Editors

The Basic Control Engine combines the power of the CIMPLICITY event handler with different scripting languages, allowing you to script and program applications and routines from the simple to the complex.

The Basic Control Engine consists of three main components.

Component	Description
Event Editor	Provides the tools to defines actions to take in response to events that occur in a process. An event can be a changing point, alarm state, or even a particular time of day. One event may invoke multiple actions, or one action may be invoked by many events.
Script Editors	Provide a set of sophisticated development tools that let you create programs. These programs can then be executed as actions in response to events. Following are the three script editors provided by CIMPLICITY: <ul data-bbox="597 1031 1198 1150" style="list-style-type: none">• CIMPLICITY Program Editor: A basic script editor.• CimScriptIDE Editor: An editor for .Net scripts.• Proficy Code Editor: An editor for Python scripts.
Basic Control Engine	Monitors for events and executes the configured actions. The Basic Control Engine is based on a multi-threaded design that allows the system to invoke and execute multiple Visual Basic programs concurrently.

About the Program Editor

The Program Editor provides an integrated development and debug environment.

Open the Program Editor. (on page 111)
Program Editor window components. (on page 114)
Program Editor: Edit programs. (on page 127)

Dialog Editor <i>(on page 148)</i>
Debug scripts. <i>(on page 217)</i>
Run a program. <i>(on page 240)</i>
Error messages. <i>(on page 241)</i>

Open the Program Editor

Open the Program Editor

Option 1 <i>(on page 111)</i>	Open a blank Program Editor.
Option 2 <i>(on page 113)</i>	Open the Program Editor with an existing script.

Option 1. Open a Blank Program Editor

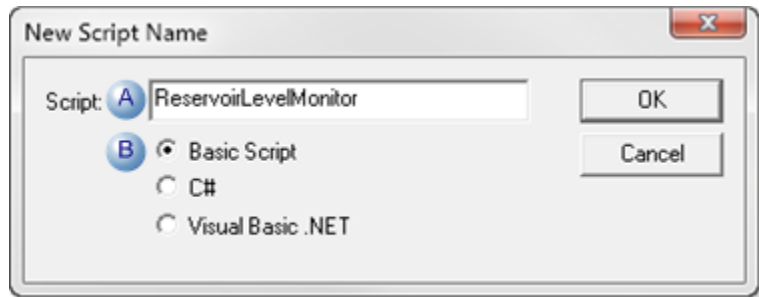
1. Select **Project>Basic Control Engine>Scripts** in the Workbench left pane.
2. Do one of the following.



A	Click File>New on the Workbench menu bar.	
B	Click the New Object button on the Workbench toolbar.	
C	In the Workbench left pane:	
	Either	Or
	Double click Scripts .	a. Right-click Scripts . b. Select New on the Popup menu.
D	a. In the Workbench right pane. a. Right-click any script (.bcl file). b. Select New on the Popup menu.	
E	Press Ctrl+N on the keyboard.	

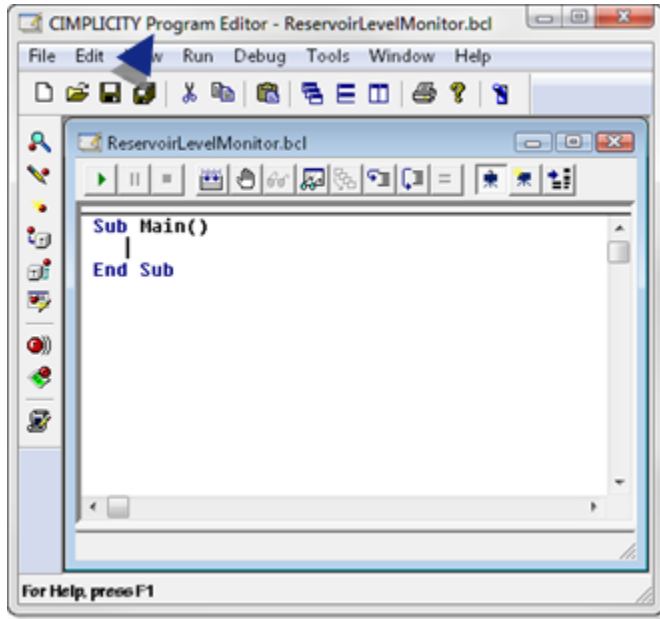
A New Script Name dialog box opens.

3. Right-click **Scripts**.
4. Select New on the Popup menu.
5. Right-click any script (.bcl file).
6. Select New on the Popup menu.
7. Do the following.



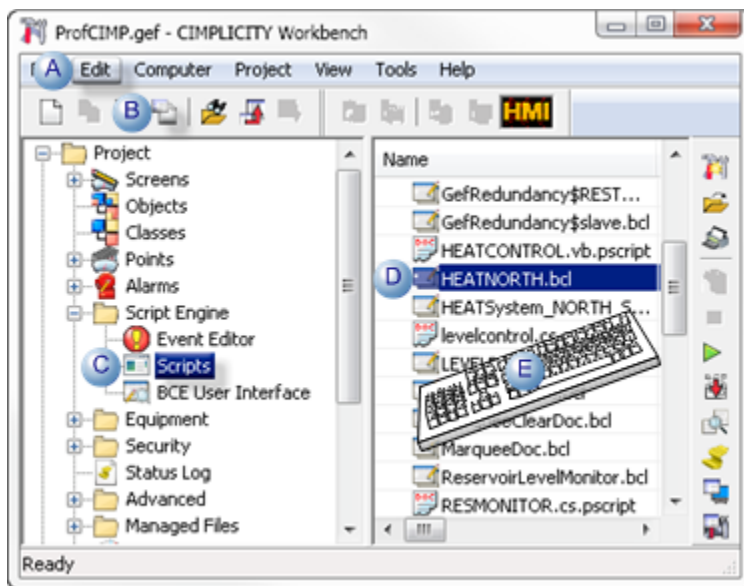
A	Enter a name in the Script field.
B	Check Basic Script.

A blank Program Editor window opens.



Option 2. Open the Program Editor with an Existing Script

1. Select **Project>Basic Control Engine>Scripts** in the Workbench left pane.
2. Select a script (.bcl file) in the Workbench right pane.
3. Do one of the following.



A	Click Edit>Properties on the Workbench menu bar.
B	Click the Properties button on the Workbench toolbar.

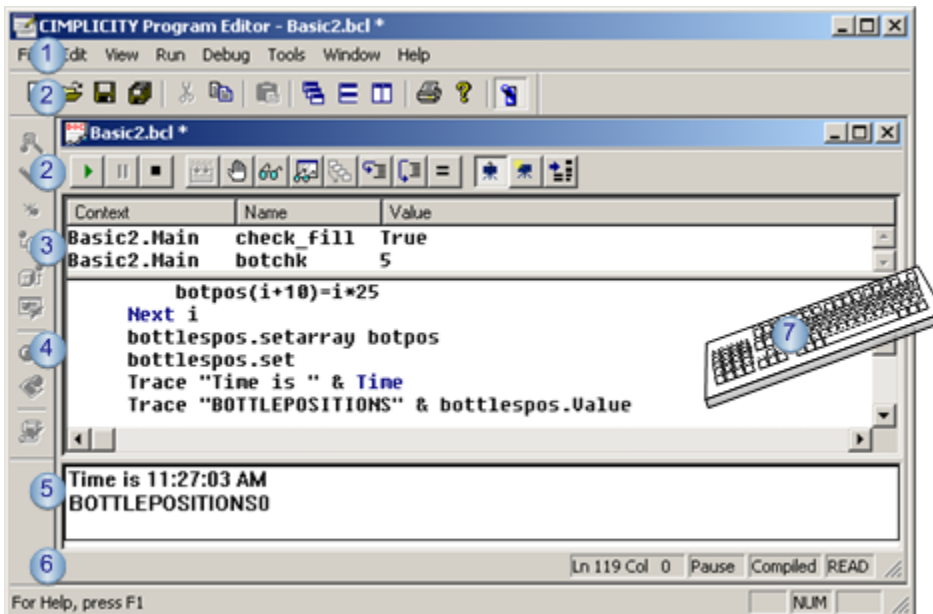
C	In the Workbench left pane: a. Right-click Scripts . b. Select Open on the Popup menu.	
D	In the Workbench right pane:	
	Either	Or
	Double click a script.	a. Right-click a script. b. Select Open on the Popup menu.
E	Press Alt+Enter on the keyboard.	

4. Right-click **Scripts**.
5. Select Open on the Popup menu.
6. Right-click a script.
7. Select Open on the Popup menu.

Program Editor Window Components

Program Editor Window Components

The Program Editor window can be divided into the following sections:



1. Program Editor Toolbars and Status Bar *(on page 121)*
2. 4. Perform Traces in Scripts *(on page 229)*
3. Scripting Language Reference *(on page 271)*
4. 5. Use a Watch Variable *(on page 232)*
5. Program Editor Toolbars and Status Bar *(on page 121)*
6. Program Editor Toolbars and Status Bar *(on page 121)*
7. Program Editor Menu Functions *(on page 116)*
8. Program Editor Shortcut Keys *(on page 124)*

1 <i>(on page 116)</i>	Menu bar
2 <i>(on page 121)</i>	Toolbars
3 <i>(on page 232)</i>	Watch area
4 <i>(on page 271)</i>	Script area
5 <i>(on page 229)</i>	Trace area
6 <i>(on page 124)</i>	Status bar
7 <i>(on page 124)</i>	Shortcut keys

**Tip:**

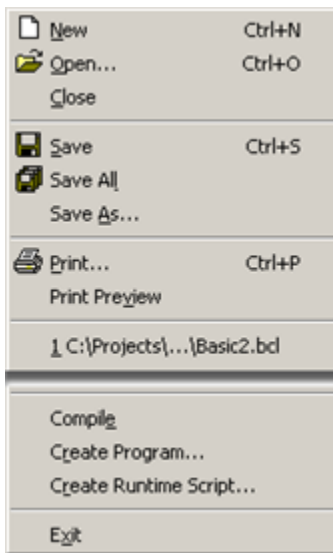
The areas can be resized by dragging the separators.

Program Editor Menu Functions

You can use the menu options to open, close, print, and compile files, to edit a file, to run a file, to debug a file, to access tools, to view status and toolbars, to arrange windows, and access help.

- File menu
- Edit menu
- Run menu
- Debug menu
- Tools menu
- View menu
- Window menu
- Help menu

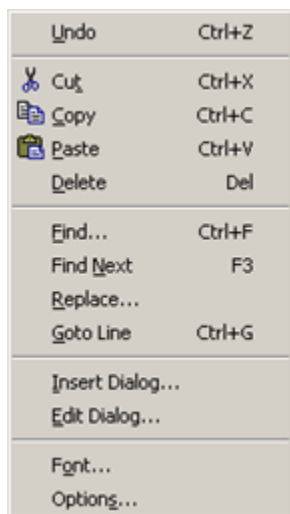
File Menu



New	Creates a new document for the Program Editor.
Open	Opens an existing document for the Program editor.
Close	Closes the script.
Save	Saves the active document.
Save All	Saves all the open files in the Program Editor.
Save As	Save the script with a different name.

Print	Prints the active document
Print Preview	Displays the active document as it will be printed
Recent Files	Displays the list of most recently accessed files.
Compile	Compiles the active document.
Create Program	Creates a new document for the Program Editor.
Exit	Exits the Program Editor.

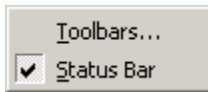
Edit Menu



Undo	Undoes the last action.
Cut	Cuts the selection and puts it on the Clipboard.
Copy	Copies the selection and puts it on the Clipboard
Paste	Inserts Clipboard contents.
Delete	Deletes the selection.
Find	Finds user-identified text in the document.
Find Next	Finds next occurrence of user-identified text in the document.
Replace	Replaces user-identified text with new text.
Goto Line	Opens a Goto Line dialog box. The insertion point goes to the line number that is entered in the Line Number field.

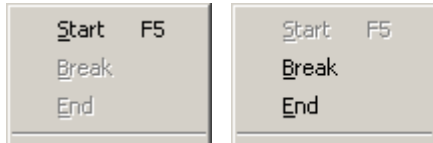
Insert Dialog	Inserts a dialog box.
Edit Dialog	Edits an inserted dialog box.
Font	Selects a font.
Options	Sets string and stack space.

View Menu



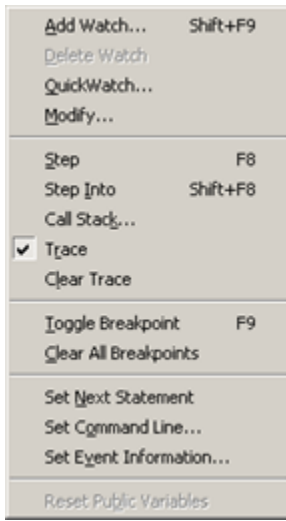
Toolbars	Displays the list of available toolbars. You can toggle the display of each toolbar.
Status Bar	Toggles the display of the Status Bar at the bottom of program windows.

Run Menu



Start	Run the program
Break	Break executing program
End	End the running or paused program

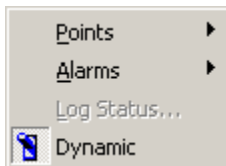
Debug Menu



Add Watch	Displays the Add Watch dialog box, in which you can specify the name of a script variable
Delete Watch	Deletes the watch from the selected variable
Quick Watch	Do a quick check of a variable value, without adding the variable to the Watch list.
Modify	Modifies the value of a variable.
Step	Executes the next line of the script. If the line calls a procedure, the called procedure is run in its entirety.
Step Into	Executes the next line of the script. If the line calls a procedure, the next line to execute will be the first line of the called procedure.
Call Stack	Displays the stack of current calls.
Trace/ Clear Trace	Enables/disables output to the Trace window
Toggle Breakpoint	Toggles a breakpoint in the script

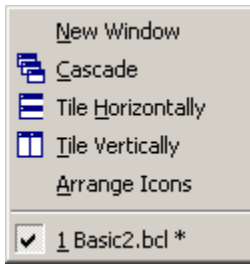
Clear all Break-points	Clears all breakpoints from the script
Set Next statement	Sets the next statement to be executed in a paused program to the currently selected line.
Set Command Line	Set the command line for the script. This can be retrieved via the basic Command\$ parameter. The Basic Control Engine will pass the Event & Action which caused the script to be run. See BCE Manual
Set Event Information	
Reset Public Variables	Re-set's the contents of public and private variables to an empty state.

Tools Menu



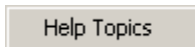
Points	Displays a submenu that lets you browse for points, edit a point, and create a new point. You can also use this menu item to include Setpoints, Get Points, and create local variables in the program.
Alarms	Displays a submenu that lets you generate or update alarms in the program.
Log Status	Displays a dialog box that lets you generate messages for the Status Log.
Dynamic	Toggles Dynamic Configuration of points, alarm, etc.

Window Menu



New Window	Opens a new window.
Cascade	Arranges the windows so that they overlap.
Tile Horizontally	Tiles the windows horizontally.
Tile Vertically	Tiles the windows vertically.
Arrange Icons	Arranges the program icons in the Program Editor window.
Current Programs	Displays the list of current programs.

Help Menu



Index	Displays the main Help window for the Program Editor.
Using Help	Displays the main Help window for Microsoft Windows.
About...	Displays program information, version number, and copyright for the Program Editor

Program Editor Toolbars and Status Bar

The Program Editor contains the following toolbars.

Toolbar	Window
Standard	Main
Tool	Main

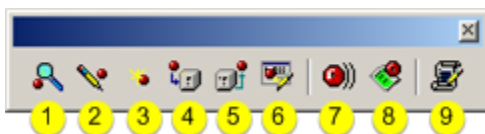
Application	Script
Status bar	Main

Standard Toolbar



1	New	Create a new document.
2	Open	Open an existing document
3	Save	Save the active document
4	Save All	Save all the open files
5	Cut	Cut the selection and put it on the Clipboard
6	Copy	Copy the selection and put it on the Clipboard
7	Paste	Insert Clipboard contents
8	Cascade Windows	Arrange windows so they overlap
9	Tile Horizontally	Arrange windows as non-overlapping tiles
10	Tile Vertically	Arrange windows as non-overlapping tiles
11	Print	Print the active document
12	About	Display program information, version number, and copy-right
13	Dynamic	Toggle Dynamic Configuration

Tools Toolbar



1	Browse Point	Browse for Points
2	Edit Point	Edit Point ID

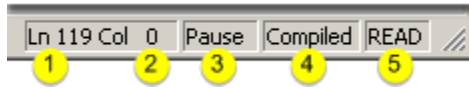
3	New Point	Create a new Point
4	Get Point	Get Point Value
5	Set Point	Set a Point
6	Dim Point	Dimension a Point Object
7	Gen Alarm	Generate an Alarm
8	Update Alarm	Update an Alarm
9	Log Status	Log a status message

Application Toolbar



1	Start	Start or continue execution
2	Break	Interrupt execution
3	End	End execution
4	Compile	Compile the document
5	Toggle Break-point	Set or clear a breakpoint
6	QuickWatch	QuickWatch a variable
7	Add Watch	Add a watch to a variable
8	Call Stack	Display call stack
9	Step Into	Step into the current line
10	Step	Step over the current line
11	Modify	Modify the value of a variable
12	Toggle Trace	Enable/Disable Tracing
13	Clear Trace	Clear the contents of the trace window
14	Command Line	Set the command line for the script

Status bar



1	Ln	Line in the script that the insertion point is on
2	Col	Column in the script that the insertion point is in
3	Pause	Script run/pause/stop status
4	Compiled	Displays if the script does not need to be compiled.
5	READ	Displays if the script is read-only.

Program Editor Shortcut Keys

The Program Editor provides several shortcut keys.

Shortcut key	Description
Ctrl+N	Creates a new document.
Ctrl+O	Opens an existing document.
Alt+F+C	Closes the active script.
Alt+F+A	Opens a Save as dialog box.
Ctrl+S	Saves the active document.
Alt+F+V	Opens a print preview window for the active script.
Ctrl+P	Prints the active document.
Ctrl+Z	Undoes the last edit action.
Ctrl+X	Cuts the selection and puts it on the Clipboard.
Ctrl+C	Copies the selection and puts it on the Clipboard.
Ctrl+V	Inserts the contents of the Clipboard.
Delete	Deletes the selection.
Ctrl+F	Opens the Find dialog box.

F3	Finds the next occurrence of the string in the Find dialog box.
Alt+E+R	Opens the Replace dialog box.
Ctrl+G	Opens the Go To Line dialog box.
Alt+E+I	Opens the Dialog Editor.
Alt+E+O	Opens the Font dialog box.
Alt+E+S	Opens the Options dialog box.
Shift+F9	Opens the Add Watch dialog box.
Alt+D+D	Delete Watch.
Alt+D+M	Modify Watch.
Alt+D+R	Trace.
Alt+D+L	Clear Trace.
F8	Steps to the next line in the script.
Shift+F8	Steps to the next line in the script. If the section is a procedure call, the next line is the first line in the procedure.
F9	Toggles a breakpoint for the debugger.
Alt+D+C	Clears all breakpoints.
Alt+D+N	Sets the next statement.
Alt+D+O	Sets the command line.
Alt+D+V	Sets event information.
Alt+D+B	Resets public variables.
F5	Starts running a script.
Alt+T+P	Opens the Points extended menu.
Alt+T+A	Opens the Alarms extended menu.
Alt+T+L	Opens the Log Status dialog box.

Set String and Stack Space

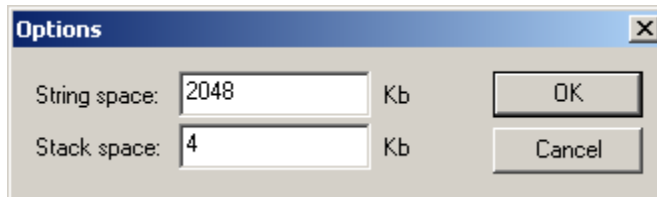
The Basic Control Engine has two regions of memory.

Memory Region	Description	
String space	Holds all string variables, arrays, and public data.	
	Default String Space size	1 MB (1024 KB)
Stack space	Holds all local variables and intermediate values.	
	Default Stack Space size	4 KB

You can change the size of either of these spaces.

The changes apply to all Basic Control Engine scripts that run as executables within the Program Editor or from the Event Manager.

1. Select Edit>Options on the Program Editor menu bar.
2. The Options dialog opens.
3. Enter the following.



Field	Description
String space	Number of kilobytes of String Space
Stack space	Number of kilobytes of Stack Space

4. Click OK.

A message opens as follows.

You must stop and restart CIMPLICITY for the changes to take effect.

5. Click OK.

**Note:**

- You can substantially reduce your stack usage by explicitly defining the types of variables (in other words, don't use variants).
- Recursive routines have an impact on Stack Space.
- If you use arrays or arrays of User Defined Types, allocation occurs in the String Space that may alleviate some stack usage.

Program Editor: Edit Programs

Program Editor: Edit Programs

Although, in some respects, editing code with the Program Editor is like editing regular text with a word-processing program, the Program Editor also has certain capabilities specifically designed to help you edit your code.

This section describes how to move around within your script, select and edit text, add comments to your script, break long statements across multiple lines, search for and replace selected text, and perform a syntax check of your script. The section ends with a brief discussion of editing dialog box templates.

1 (on page 128)	Program Editor: Navigate within a script.
2 (on page 129)	Program Editor: Text procedures.
3 (on page 137)	Program Editor: Point tools.
4 (on page 143)	Program Editor: Alarm tools.
5 (on page 145)	Program Editor: Log status tool.

6 (on page 146)	Program Editor: Add comments to a script
7 (on page 147)	Program Editor: Enter a statement across multiple lines.
8 (on page 147)	Program Editor: Check the syntax of a script.
9 (on page 148)	Program Editor: Add dialog boxes to a script.

1. Program Editor: Navigate within a Script

When you move the insertion point with a [keyboard shortcut](#), ([on page 124](#)) the Program Editor scrolls the new location of the insertion point into view if it is not already displayed.

You can also reposition the insertion point with the mouse and the Goto Line command.

- Move the insertion point with the mouse.
- Move the Insertion point to a specified line.



Note:

The Program Editor allows you to place the insertion point anywhere within your script, including in "empty spaces." (Empty spaces are areas within the script that do not contain text, such as a tabs expanded space or the area beyond the last character on a line.)

Move the Insertion Point with the Mouse

1. Use the scroll bars at the right and bottom of the display to scroll the target area of the script into view if it is not already visible.
2. Place the cursor where you want to position the insertion point.
3. Click the left mouse button.

Result: The insertion point is repositioned.

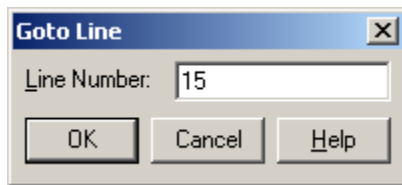
**Note:**

- This approach is especially fast if the area of the screen to which you want to move the insertion point is currently visible.
- When you scroll the display with the mouse, the insertion point remains in its original position until you reposition it with a mouse click. If you attempt to perform an editing operation when the insertion point is not in view, Program Editor automatically scrolls the insertion point into view before performing the operation.

Move the Insertion Point to a Specified Line

4. Press F4.

Program Editor displays the Goto Line dialog box.



5. Enter the number of the line in your script to which you want to move the insertion point.
6. Click the OK button or press Enter.

The insertion point is positioned at the start of the line you specified. If that line was not already displayed, the Program Editor scrolls it into view.

**Note:**

- This approach is especially fast if the area of the screen to which you want to move the insertion point is not currently visible but you know the number of the target line.
- The insertion point cannot be moved so far below the end of a script as to scroll the script entirely off the display. When the last line of your script becomes the first line on your screen, the script will stop scrolling, and you will be unable to move the insertion point below the bottom of that screen.

2. Program Editor: Text Procedures

2. Program Editor: Text Procedures

2.1 (on page 130)	Insert text.
2.2 (on page 131)	Select/delete text.
2.3 (on page 133)	Cut/copy/paste text.
2.4 (on page 134)	Undo editing operations.
2.5 (on page 135)	Search/replace text.

2.1. Insert Text

Position the insertion point at the desired location in the script and start typing.



guide:

Guidelines

- Text does not wrap. If you continue typing on a given line, eventually you will reach a point at which you can enter no more text on that line.
- Press **Enter** to control the line breaks when you want to insert a new line in your script.

The effect of pressing Enter depends on where the insertion point is located at the time:

Insertion point location	When Enter is pressed
At or beyond the end of a line	A new line is inserted after the current line.
At the start of a line	A new line is inserted before the current line.

Within a line	The current line is broken into two lines at that location.
---------------	---

- Press Tab to insert a tab character

The tab character is inserted at the location of the insertion point, which causes text after the tab to be moved to the next tab position.

If you insert new text within a tab's expanded space, the text that originally appeared on that line is moved to the next tab position each time the new text that you are entering reaches the start of another tab position.



Tip:

Because you can position the insertion point in empty spaces, you can also insert text in empty spaces. This is useful for inserting a [comment \(on page 146\)](#) in the space beyond the end of a line in the script.

When you insert characters beyond the end of a line, the space between the insertion point and the last character on the line is back filled with tab characters.

2.2. Select/Delete Text

- Select text.
- Delete text.

Select text

You can use either the mouse or the keyboard to select text and other characters in your script.



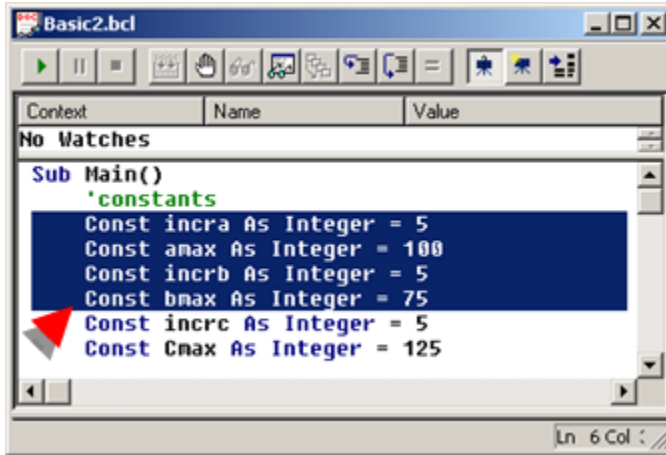
Important:

Regardless of which method you use, you can select either a portion of one line or a series of whole lines, but you cannot select a portion of one line plus one or more whole lines.

When you select multiple lines and start or end your selection partially through a line, the Program Editor automatically extends the selection to include the entire starting and ending lines.

Options for selecting text include:

- Text with the mouse.
- Text with the keyboard.
- Line with the keyboard.



Text with the Mouse

1. Place the insertion point where you want your selection to begin
2. Do one of the following.
 - While pressing the left-mouse button
 - a. Drag the mouse until you reach the end of your selection.
 - b. Release the mouse button.
 - Using the left-mouse button.
 - a. Place the mouse pointer in the left margin beside the first line you want to select.
 - b. The pointer becomes a reverse arrow, which points toward the line of text.
 - c. Click the left mouse button to select a single line.
 - d. Press the left mouse button and drag up or down to select multiple lines.
 - While pressing **Shift**
 - a. Place the mouse pointer where you want your selection to end
 - b. Click the left mouse button.

Result: The selected text is highlighted on your display.

Text With the Keyboard

3. Place the insertion point where you want your selection to begin.
4. While pressing **Shift**, use one of the navigating keyboard shortcuts to extend the selection to the desired ending point.

Result: The selected text is highlighted on your display.

**Note:**

When you intend to select an entire single line of text in your script, it is important to remember to extend your selection far enough to include the hidden end-of-line character, which is the character that inserts a new line in your script.

Line With the Keyboard

5. Place the insertion point at the beginning of the line you want to select.
6. Press Shift + Down arrow.

The entire line, including the end-of-line character, is selected.

7. Repeat 2 to extend your selection to include additional whole lines of text.
8. Place the insertion point in that line.
9. Press Ctrl+Y.
10. Place the insertion point after the last character on the current line.
11. Press Delete once to delete the hidden end-of-line character.
12. Place the insertion point at the start of a line.
13. Press Backspace.

2.3. Cut/Copy/Paste Text

Place material from your script on the Clipboard by either cutting it or copying it.

- Cut a selection.
- Copy a selection.
- Paste text.

Cut a selection

1. [Select \(on page 131\)](#) the text to cut.
2. Do one of the following.
 - Press Ctrl+X.
 - Click the [Cut \(on page 122\)](#) button on the Program Editor toolbar.
 - Click Edit>Cut on the Program Editor menu bar.

Result: The selection is cut from the script and placed on the Clipboard.

Copy a selection

3. [Select \(on page 131\)](#) the text to copy.
4. Do one of the following.
 - Press Ctrl+C.
 - Click the [Copy \(on page 122\)](#) button on the Program Editor toolbar.
 - Click Edit>Copy on the Program Editor menu bar.

Result: The selection remains in the script, and a copy of it is placed on the Clipboard.

Paste text

5. Position the insertion point where the copied or cut text should be placed.
6. Do one of the following.
 - Press Ctrl+V.
 - Click the [Paste \(on page 122\)](#) button on the Program Editor toolbar.
 - Click Edit>Paste on the Program Editor menu bar.

The contents of the Clipboard are pasted at the insertion point location.



Note:

If text is selected when you paste Clipboard text, the Clipboard text is inserted before the selected text.

2.4. Undo Editing Operations

Any of the following editing operations that produce a change in the script can be undone.

- Insertion of a series of characters
- Insertion of a block of text from the Clipboard
- Deletion of a series of characters
- Deletion or cutting of a block of text



Important:

You can undo the last 100 operations.

Do either of the following to undo an editing operation.

- Press Ctrl+Z
- Click Edit>Undo on the Program Editor menu bar.

Result: The script is restored to the way it looked before you performed the editing operation.



Note:

Operations that do not produce any change in the script and cannot be undone include:

- Moving the insertion point
- Selecting text
- Copying material to the Clipboard.

2.5. Search/Replace Text

Program Editor makes it easy to search for specified text in your script and automatically replace instances of specified text.

- Find text in your script.
- Replace text in your script.

Find text in a script

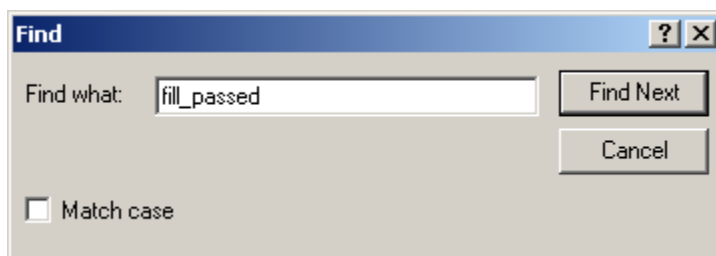
1. Move the insertion point to where the search will start.

Tip: To start at the beginning of the script, press Ctrl+Home.

2. Do either of the following.
 - Press Ctrl+F.
 - Click Edit>Find on the Program Editor menu bar.

A Find dialog box opens.

3. Do the following.



Feature	Description
---------	-------------

Find what	Enter the text to find.
Match case	Check to limit the results to the case entered in the Find what field.
Find Next	Click to start the search and continue the search when a match is found. Tip: You can also press Enter.

Results

- The Find dialog box remains open
 - The Program Editor either highlights the first instance of the specified text or reports that the text cannot be found.
4. If the specified text has been found, click Find Next or press Enter to search for the next instance.



Tip:

If the Find dialog box blocks your view of an instance of the specified text, you can do either of the following.

- Move the dialog box out of your way and continue with your search.
- Click Cancel.

The Find dialog box closes, but maintains the established search criteria.

Press F3 to find successive occurrences of the specified text without re-opening the Find dialog box.

Note: If you press F3 when there is no entry in the **Find what** field the Program Editor opens the Find dialog box.

Automatically replace text in a script

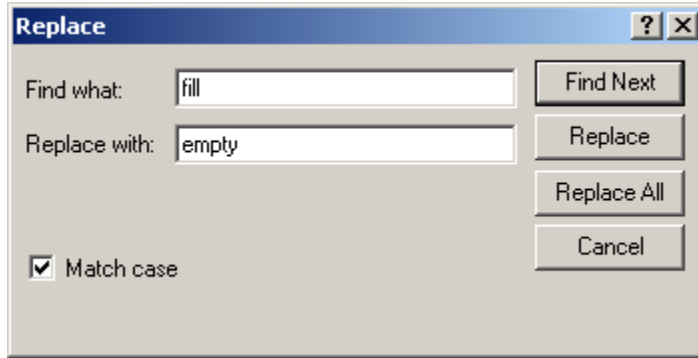
5. Move the insertion point to where you the replacement operation should start.

Tip: To start at the beginning of the script, press Ctrl+Home.

6. Do either of the following.
- Press Alt+E+R on the keyboard.
 - Click Edit>Replace on the Program Editor menu bar.

The Replace dialog box opens.

7. Do the following.



Feature	Description
Find what	Enter the text to find.
Replace with	Enter the replacement text.
Match case	Check to limit the results to the case entered in the Find what field.
Find Next	Click to start the search and continue the search when a match is found. Tip: You can also press Enter.
Replace	Click to replace a found instance of the text. Result: When the text is replaced either of the following happens. <ul style="list-style-type: none"> ◦ The cursor highlights the next found instance ◦ A message reports that there are no more instances of the Find what text.
Replace All	(Optional) Click to automatically replace all found instances of the Find what text with the replacement text. Result: All instances are replaced with no requests for confirmation.
Cancel	Click to cancel the Find/Replace operation.

3. Program Editor: Point Tools

The Program Editor provides tools to facilitate working with points in a script, as follows.



1. [#unique_73_Connect_42_i6Dim \(on page 142\)](#)
2. [#unique_73_Connect_42_i5Get \(on page 141\)](#)
3. [#unique_73_Connect_42_i4Set \(on page 141\)](#)
4. [#unique_73_Connect_42_i3New \(on page 140\)](#)
5. [#unique_73_Connect_42_i2Edit \(on page 139\)](#)
6. [#unique_73_Connect_42_i1Browse \(on page 138\)](#)

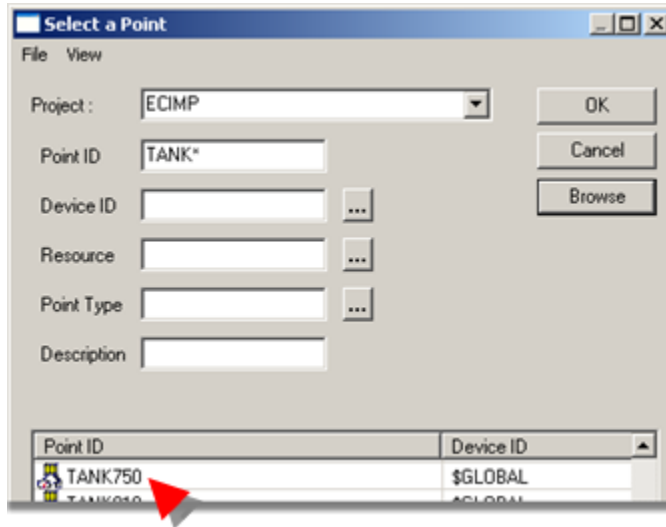
1 (on page 138)	Browse
2 (on page 139)	Edit
3 (on page 140)	New
4 (on page 141)	Set
5 (on page 141)	Get
6 (on page 142)	Dim

1	Browse
---	--------

1. Place the insertion point in the script where the selected point will be inserted.
2. Click Tools>Points>Browse on the Program Editor menu bar.

The Select a Point Browser opens.

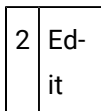
3. Select the point to be inserted.



4. Click OK.

Result: The point ID is inserted in double quotes at the insertion point.

```
Dim x as new Point
x.Id = "TANK750"
x.Get
"TANK750"
```

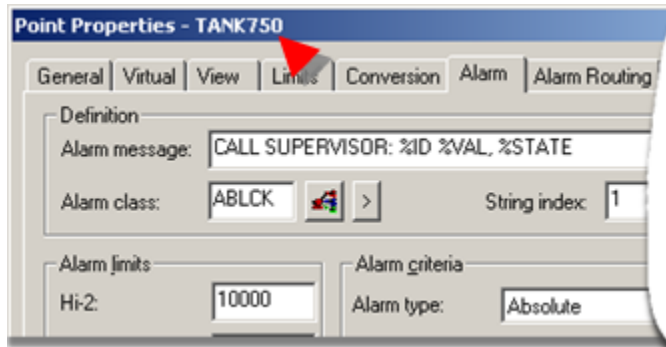


5. Select a point ID in the script.

```
Dim x as new Point
x.Id = "TANK750"
x.Get
```

6. Click Tools>Points>Edit on the Program Editor menu bar.

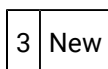
The Point Properties dialog box opens for the selected point.



7. Make required edits to the point.
8. Close the Point Properties dialog box.

Result: The selected point configuration is edited if the project is:

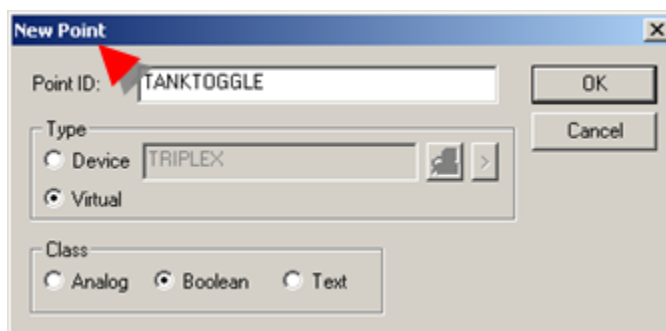
- Running and Dynamic Configuration is enabled
- Not running after a configuration update has been performed.



9. Place the insertion point in the script where the new point will be inserted.
10. Click Tools>Points>New on the Program Editor menu bar.

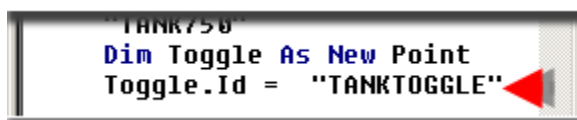
A New Point dialog box opens.

11. Create and configure a new point (in the Point Properties dialog box).



12. Close the point's Point Properties dialog box.

Result: The new point ID is inserted in the script at the insertion point.

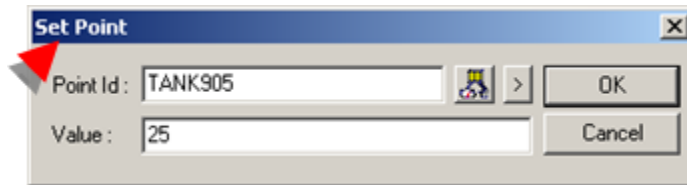


4	Set
---	-----

- Place the insertion point in the script where the `PointSet` (on page 913) statement will be inserted.
- Click Tools>Points>Set on the Program Editor menu bar.

A Set Point dialog box opens.

- Fill in the fields as follows.



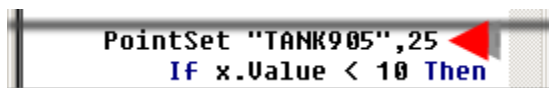
Field	Description
Point ID	Point ID that will be inserted in the <code>PointSet</code> (on page 913) statement.
Value	Value assigned to the point in the <code>PointSet</code> (on page 913) statement.

- Click OK.

Result: A `PointSet` statement is inserted at the insertion point in the script

Example

```
PointSet "TANK905" ,25
```



5	Get
---	-----

- Place the insertion point in the script where the `PointGet` (on page 906) function will be inserted.
- Click Tools>Points>Get on the Program Editor menu bar.

A Get Point dialog box opens.

- Fill in the fields as follows.



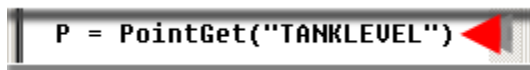
Field	Description
Point ID	Point ID that will be inserted in the <code>PointGet (on page 906)</code> function.
Returns	Value received by the point in the <code>PointGet (on page 906)</code> function.

20. Click OK.

Result: A PointGet function is inserted at the insertion point in the script.

Example

```
P = PointGet ("TANKLEVEL")
```



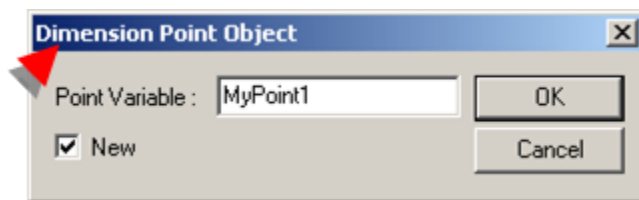
6	Dim
---	-----

21. Place the insertion point in the script where a `Dim (on page 426)` statement will be inserted.

22. Click Tools>Points>Dim on the Program Editor menu bar.

A Dimension Point Object dialog box opens.

23. Fill in the fields as follows.



Field	Description
-------	-------------

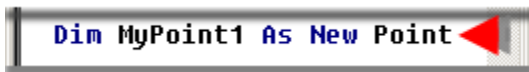
Point Variable	Point variable ID
New	Check to declare a new instance of the point variable.

24. Click OK.

A Dim statement for a point or new point variable is inserted at the insertion point in the script.

Example

```
Dim MyPoint1 As New Point
```



4. Program Editor: Alarm Tools

The Program Editor provides tools to facilitate working with alarms in a script, as follows.



1. [#unique_74_Connect_42_i1Generate \(on page 143\)](#)
2. [#unique_74_Connect_42_i2Update \(on page 144\)](#)

1	Gener- ate
---	---------------

The Generate tool in the Program Editor is available for \$CIMBASIC alarms.

1. Place the insertion point in the script where the selected alarm will be inserted.
2. Click Tools>Alarms>Generate on the Program Editor menu bar.

A Generate Alarm dialog box opens.

1. Fill in the fields for a \$CIMBASIC alarm in the Generate Alarm dialog box.

Note: The fields in the Generate Alarm dialog box correspond to the [AlarmGenerate \(on page 806\)](#) (method) syntax.

1. Click OK.

Result: The Program Editor inserts the specified [AlarmGenerate \(on page 806\)](#) (method) code at the location of the insertion point in the script.

```
ALarmGenerate "ECIMP","BASIC","$SYSTEM","CALL THE TANK
```

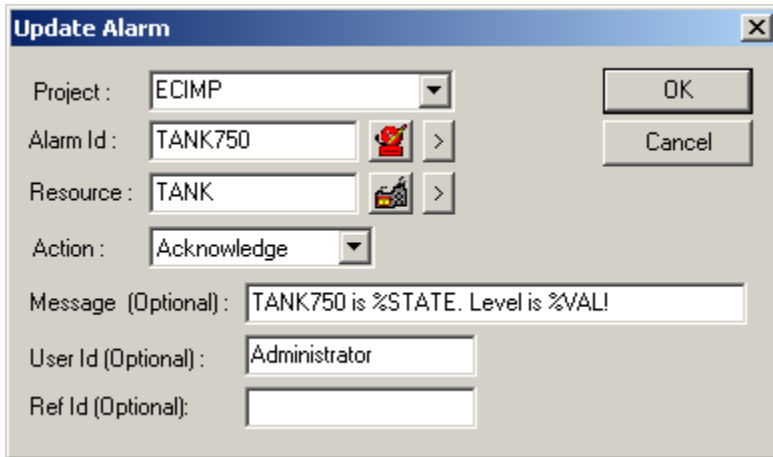
2	Update
---	--------

The Update tool in the Program Editor is available for any CIMPLICITY alarm.

1. Place the insertion point in the script where the selected alarm will be inserted.
2. Click Tools>Alarms>Update on the Program Editor menu bar.

An Update Alarm dialog box opens.

1. Fill in the fields in the Update Alarm dialog box.



The 'Update Alarm' dialog box contains the following fields and controls:

- Project: ECIMP (dropdown menu)
- Alarm Id: TANK750 (text field with a fire alarm icon and a right arrow)
- Resource: TANK (text field with a tank icon and a right arrow)
- Action: Acknowledge (dropdown menu)
- Message (Optional): TANK750 is %STATE. Level is %VAL! (text field)
- User Id (Optional): Administrator (text field)
- Ref Id (Optional): (empty text field)
- Buttons: OK and Cancel

Note: The fields in the Update Alarm dialog box correspond to the `AlarmUpdate` (on page 813) (method) syntax.

1. Click OK.

Result: The Program Editor inserts the specified `AlarmUpdate` (on page 813) (method) code at the location of the insertion point in the script.

```
AlarmUpdate "ECIMP","TANK750","TANK",AM_ACKNOWLEDGED,""
```

5. Program Editor: Log Status Tool

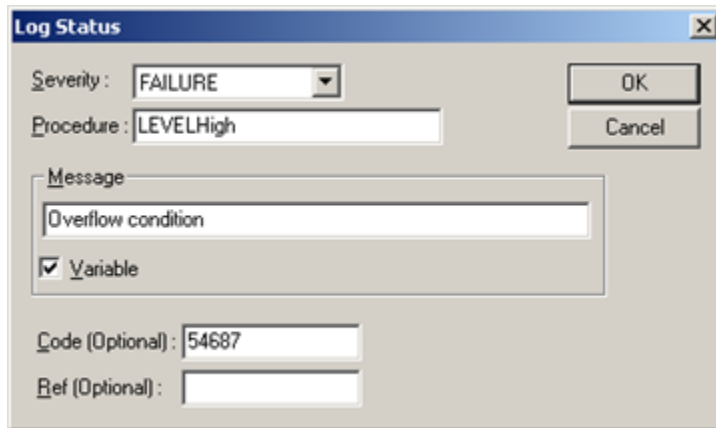
1. Click Tools>Log Status on the Program Editor menu bar.



A Log Status dialog box opens.

2. Fill in the fields to specify log status criteria.

The fields correspond to the `LogStatus` (on page 866) (property, read/write) syntax.



6. Program Editor: Add Comments to a Script

You can add comments to your script to remind yourself or others of how your code works. Comments are ignored when your script is executed.

The apostrophe symbol (') is used to indicate that the text from the apostrophe to the end of the line is a comment.

Add a:

- Full line comment
- End of line comment.

Full line comment

1. Type an apostrophe (') at the start of the line.
2. Type a comment following the apostrophe.

Result: When the script is run, the presence of the apostrophe at the start of the line will cause the entire line to be ignored.

End of line comment

3. Position the insertion point in the empty space beyond the end of the line of code.
4. Type an apostrophe (').
5. Type a comment following the apostrophe.

When the script is run, the code on the first portion of the line will be executed, but the presence of the apostrophe at the start of the comment will cause the remainder of the line to be ignored.



Note:

Although you can place a comment at the end of a line containing executable code, you cannot place executable code at the end of a line containing a comment; the presence of the apostrophe at the start of the comment will cause the balance of the line (including the code) to be ignored.

7. Program Editor: Enter a Statement across Multiple Lines

1. Type the statement on multiple lines, exactly the way you want it to appear.
2. Place the insertion point at the end of the first line in the series.
3. Press the spacebar once to insert a single space.
4. Type an underscore (_).

Note: The underscore is the line-continuation character, which indicates that the statement continues on the following line.

5. Repeat 2 - 4 to place a line-continuation character at the end of each line in the series, except the last.

8. Program Editor: Check the Syntax of a Script

1. Do one of the following.
 - Click [Compile \(on page 123\)](#) on the Application toolbar.
 - Click File>Compile on the Program Editor menu bar.

The Program Editor does one of the following.

- Reports that no errors have been found
 - Displays an error message that specifies the first line in your script where an error has been found and briefly describes the nature of that error.
2. Click the OK button or press Enter on the keyboard.

If Program Editor has found a syntax error, the line containing the error is highlighted on your display.

3. Correct the syntax error.
4. Repeat the procedure until you have found and corrected all syntax errors.

9. Program Editor: Add Dialog Boxes to a Script

Basic Control Engine syntax provides several options for adding dialog boxes to a script.

The Program Editor enables you to do either of the following.

Write a script beginning with <code>Begin Dialog</code> to add a dialog box to a script.
Use the Dialog Editor.

Dialog Editor

Dialog Editor

You can use a custom dialog box to display information to a user while providing an opportunity for the user to respond. The Dialog Editor is a tool that enables you to create and modify custom dialog boxes for use in your scripts. Although the statements used to display a custom dialog box and respond to the choices made by a user of the dialog box may seem complicated, the Dialog Editor makes it easy to generate these statements.

This chapter contains the following topics:

1 (on page 149)	Use the Dialog Editor.
2 (on page 156)	Create a custom dialog box.
3 (on page 164)	Edit a custom dialog box.
4 (on page 191)	Insert/paste a dialog box template code into a script.
5 (on page 195)	Edit an existing dialog box.

6 (on page 202)	Test a dialog box.
7 (on page 206)	Exit from the Dialog Editor.
8 (on page 208)	Use a custom dialog box in a script.
9 (on page 213)	Use a dynamic dialog box in a script.

1. Use the Dialog Editor

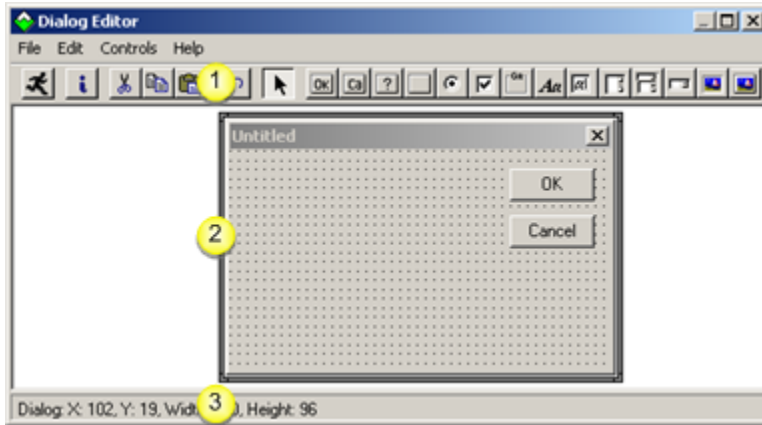
1. Use the Dialog Editor

1.1 (on page 149)	Dialog Editor application window.
1.2 (on page 150)	Dialog Editor toolbar
1.3 (on page 151)	Dialog Editor menus
1.4 (on page 155)	Keyboard shortcuts for the Dialog Editor.

1.1. Dialog Editor Application Window

Dialog boxes created with Dialog Editor normally appear in an 8 point Helvetica font, both in Dialog Editor's application window and when the corresponding code is run.

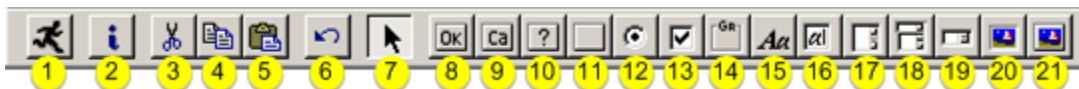
The application window contains the following elements.



Feature	Description
1 Toolbar	Collection of tools that you can use to provide instructions to the Dialog Editor, as discussed in the following subsection
2 Dialog box	Visual layout of the dialog box that you are currently creating or editing.
3 Status bar	Provides key information about the operation you are currently performing, including the name of the currently selected control or dialog box, together with its position on the display and its dimensions; the name of a control you are about to add to the dialog box with the mouse pointer, together with the pointer's position on the display; the function of the currently selected menu command; and the activation of Dialog Editor's testing or capturing functions.

1.2. Dialog Editor Toolbar

Buttons on the Dialog Editor's toolbar are as follows.



Tool	Function
1 Run	Runs the dialog box for testing purposes.
2 Information	Displays the Information dialog box for the selected dialog box or control.

3	Cut	Cuts a selection; places the contents on the Clipboard.
4	Copy	Copies a selection; places the contents on the Clipboard.
5	Paste	Pastes Clipboard contents.
6	Undo	Undoes the last action.
7	Select	Lets you select, move, and resize items and control the insertion point.
8	OK Button	Adds an OK button to the dialog box.
9	Cancel Button	Adds a Cancel button to the dialog box.
10	Help Button	Adds a Help button to the dialog box.
11	Push Button	Adds a push button to the dialog box.
12	Option Button	Adds an option button to the dialog box.
13	Check Box	Adds a check box to the dialog box.
14	Group Box	Adds a group box to the dialog box.
15	Text	Adds a text control to the dialog box.
16	Text Box	Adds a text box to the dialog box.
17	List Box	Adds a list box to the dialog box.
18	Combo Box	Adds a combo box to the dialog box.
19	Drop List Box	Adds a drop list box to the dialog box.
20	Picture	Adds a picture to the dialog box.
21	Picture Button	Adds a picture button to the dialog box.

1.3. Dialog Editor Menus

- File menu
- Edit menu
- Controls menu
- Help menu

File Menu

New	Ctrl+N
Open...	Ctrl+O
Update	
Save As...	
Test Dialog	F5
Capture Dialog...	
Exit and Return	

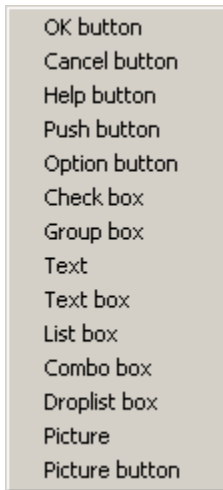
Selection	Function
New	Creates a new dialog box.
Open...	Opens the Open Dialog File dialog box, which you can use to open an existing dialog box template.
Update	Updates the template. Does one of the following in the open Program Editor script. <ul style="list-style-type: none"> • Inserts the template code, if it is not in the script. • Updates existing template code in the script.
Save As...	Opens a Save As Dialog File dialog box, which you can use to save the current dialog box template in a file under the same or new name.
Test Dialog	Toggles between: <ul style="list-style-type: none"> • Run mode in which the dialog box emulates a dialog box during runtime for testing purposes • Edit mode in which changes can be made to the dialog box.
Capture Dialog	Captures the standard Windows controls from a standard Windows dialog box in another Windows application.
Exit & Return	Closes the Dialog Editor and returns you to the host application.

Edit Menu

Undo Move	Ctrl+Z
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Clear	Delete
Duplicate	Ctrl+D
Size to text	F2
Info...	Ctrl+I
Grid...	Ctrl+G

Selection	Function
Undo	Undo up to 10 preceding operations. The Undo command continually indicates the next operation you can undo by selecting it and grays out when there are no more operations that can be undone.
Cut	Cuts the selected dialog box or control from the Dialog Editor window and places it on the Clipboard.
Copy	Copies the selected dialog box or item, without removing it from the Dialog Editor window, and places it on the Clipboard.
Paste	Inserts cut or copied dialog box or items into the Dialog Editor.
Clear	Deletes the selected dialog box or control from Dialog Editor's application window without placing it on the Clipboard.
Duplicate	Creates a duplicate copy of the selected item.
Size to Text	Adjusts the borders of certain items to fit the text displayed on them.
Info...	Displays an Information dialog box for the selected dialog box or item.
Grid...	Displays the Grid dialog box.

Controls Menu



Selection	Function
OK button	Adds a default OK button to the dialog box.
Cancel button	Adds a default Cancel button to the dialog box.
Push button	Adds a push, or command, button to the dialog box.
Option button	Adds an option button to the dialog box.
Check box	Adds a check box to the dialog box.
Group box	Adds a group box to the dialog box.
Text	Adds a text control to the dialog box.
Text box	Adds a text box to the dialog box.
List box	Adds a list box to the dialog box.
Combo box	Adds a combo box to the dialog box.
Drop list box	Adds a drop list box to the dialog box.
Picture	Adds a picture to the dialog box.
Picture button	Adds a picture button to the dialog box.

Help Menu

Help Topics

Selection	Function
Help Topics	Opens documentation for the Dialog Editor.

1.4. Keyboard Shortcuts for the Dialog Editor

The following keyboard shortcuts can be used for some of the operations you will perform most frequently in Dialog Editor.

Key(s)	Function
Alt +F4	Closes Dialog Editor's application window.
Ctrl+C	Copies the selected dialog box or control, without removing it from Dialog Editor's application window, and places it on the Clipboard.
Ctrl+D	Creates a duplicate copy of the selected control.
Ctrl+G	Displays the Grid dialog box.
Ctrl+I	Displays the Information dialog box for the selected dialog box or control.
Ctrl+V	Inserts the contents of the Clipboard into Dialog Editor. If the Clipboard contains script statements describing one or more controls, then Dialog Editor adds those controls to the current dialog box. If the Clipboard contains the script template for an entire dialog box, then Dialog Editor creates a new dialog box from the statements in the template.
Ctrl+X	Removes the selected dialog box or control from Dialog Editor's application window and places it on the Clipboard.
Ctrl+Z	Undoes the preceding operation.
Del	Removes the selected dialog box or control from Dialog Editor's application window without placing it on the Clipboard.
F1	Displays Help for the currently active window.
F2	Runs the dialog box for testing purposes.

F3	Sizes certain controls to fit the text they contain.
Shift +F1	Toggles the Help pointer.

2. Create a Custom Dialog Box

2. Create a Custom Dialog Box

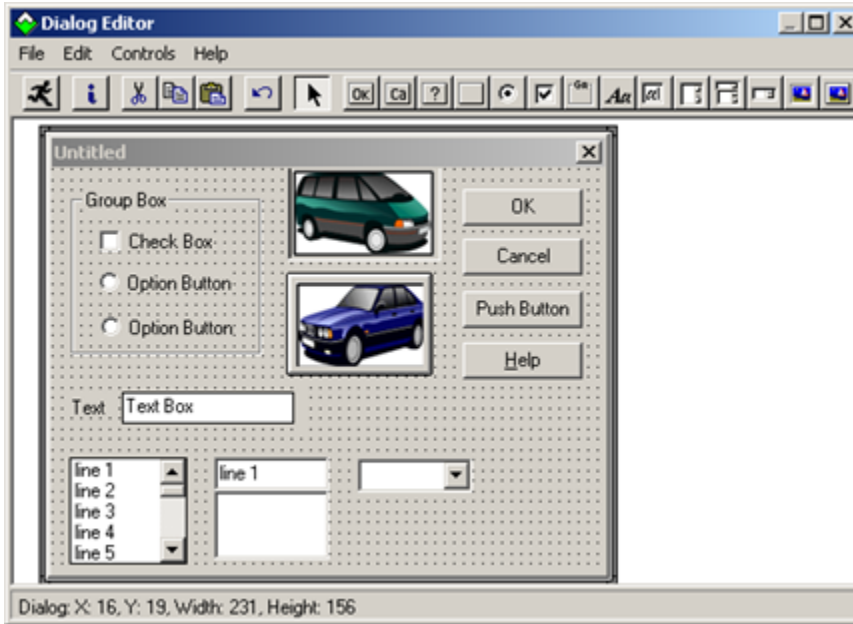
2.1 (on page 156)	Review available controls.
2.2 (on page 160)	Add controls to a dialog box.
2.3 (on page 162)	Use the grid to position controls within a dialog box
2.4 (on page 163)	Save the custom dialog box.

2.1. Review Available Controls

- Available controls
- Control guidelines

Available Controls

The Dialog Editor supports the following types of standard Windows controls, all of which are illustrated in the above dialog box:



Feature	Description
Check box	Box that users can check or clear to indicate their preference regarding the alternative specified on the check box label.
Combo box	Text field with a displayed, scroll list beneath it. Users can either select an item from the list or enter the name of the desired item in the text field. The currently selected item is displayed in the text field. If the item was selected from the scrolling list, it is highlighted there as well.
Drop list box	Field that displays the currently selected item, followed by a downward-pointing arrow, which users can click to temporarily display a scrolling list of items. Once a user selects an item in the list, the list disappears and the newly selected item displays in the field.
Group box	Rectangular design element used to enclose a group of related controls. An optional group box label is available to display a title for the controls in the box..
List box	Displayed scroll list from which users can select one item. The currently selected item is highlighted on the list.
Picture	Displays a Windows bitmap or metafile.
Picture	Push/Command, button that displays a Windows bitmap or metafile.

but- ton	
Push but- ton	Command button. Note: Push buttons include: <ul style="list-style-type: none"> • OK buttons. • Cancel buttons. • Picture buttons.
Op- tion but- ton	One of a group of two or more linked buttons that allows users select only one from a group of mutually exclusive choices. Clicking an unselected button in the group selects that button and automatically de-selects the previously selected button in that group.
Text	Read-only field that contains text the users' information.
Text box	Field into which users can enter text (potentially, as much as 32K). <ul style="list-style-type: none"> • By default, the Text box holds a single line of non-wrapping text. • The field can multiple lines of wrapping text.

Control Guidelines

- General guidelines
- Tabbing order.
- Option button grouping.
- Accelerator keys.

General guidelines

- A single dialog box can contain no more than 255 controls
- A dialog box will not operate properly unless it contains either an **OK** button, a **Cancel** button, a push button, or a picture button.

An **OK** button and a **Cancel** button are provided by default on a new dialog box.

- Group boxes, text controls, and pictures are passive elements in a dialog box, inasmuch as they are used purely for decorative or informative purposes. Users cannot act upon these controls, and when they tab through the dialog box, the focus skips over these controls.
- A Windows bitmap or metafile can be obtained from a file or from a specified library.

Tabbing order

Users can select dialog box controls by tabbing.

The order in which you create the controls is what determines the tabbing order, not the position of the controls in the dialog box.

The closer you can come to creating controls in the order in which you want them to receive the tabbing focus, the fewer tabbing-order adjustments you will have to make later on.

Option button grouping

If you want a series of option buttons to work together as a mutually exclusive group, you must create all the buttons in that group one right after the other, in an unbroken sequence.

If you create a different type of control before you have finished creating all the option buttons in your group, you will split the buttons into two or more separate groups.

Example

You plan to create an option button group with five buttons.

You create in the following order.

1. Three of the buttons
2. A list box.
3. Two buttons

When you test the dialog box, the five buttons will not work together as a mutually exclusive group.

Instead the:

- First three buttons will form one mutually exclusive group.
- Last two buttons will form another mutually exclusive group.

Accelerator keys

You can provide easy access to

- A text box, list box, combo box, or drop list box by assigning an accelerator key to an associated text control.
- The controls in a group box by assigning an accelerator key to the group box label.

To do this, you must create the text control or group box first, followed immediately by the controls that you want to associate with it. If the controls are not created in the correct order, they will not be associated in your dialog box template and any accelerator key you assign to the text control or group box label will not work properly.

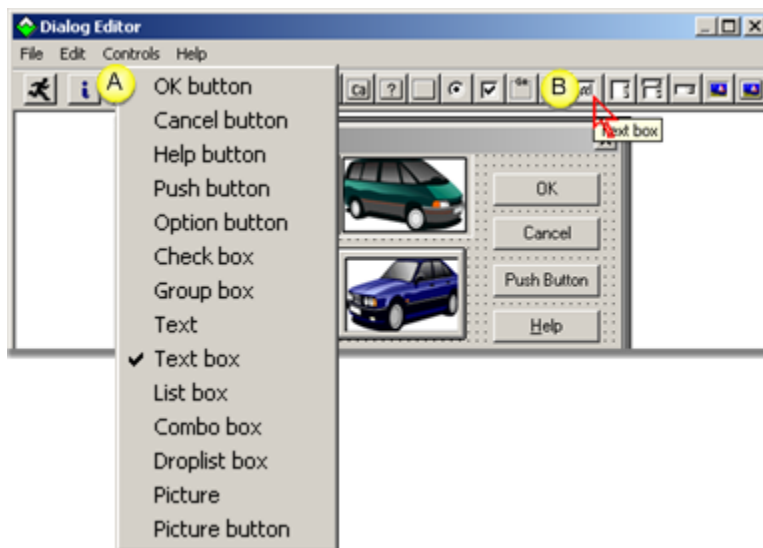
2.2. Add Controls to a Dialog Box



Note:

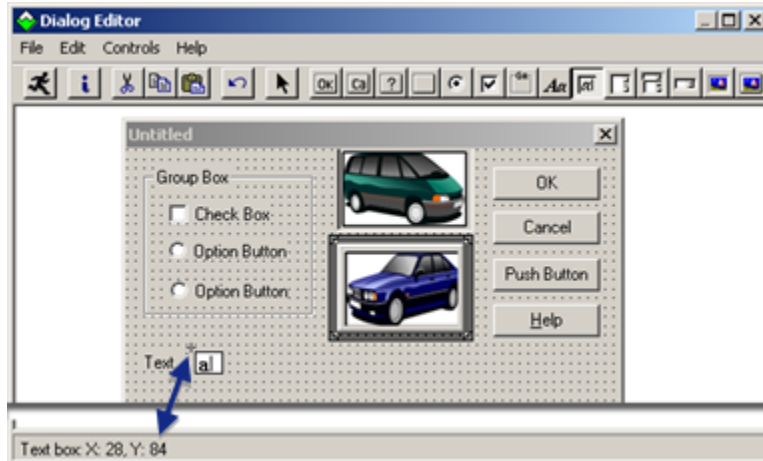
You can only insert a control within the borders of the dialog box you are creating. You cannot insert a control on the dialog box's title bar or outside its borders.

1. Do one of the following.



A	Click Controls><control object> on the Dialog Editor menu bar.
B	Click the button (on page 150) on the Dialog Editor toolbar that corresponds to the type of control you want to add.

2. Place the cursor in the dialog box where you want the upper left corner of the control to be positioned.
 - As soon as you place the cursor in the dialog box it changes to a crossbar accompanied by a small image of the selected object.
 - The Dialog Editor status bar displays the crossbar's coordinates.



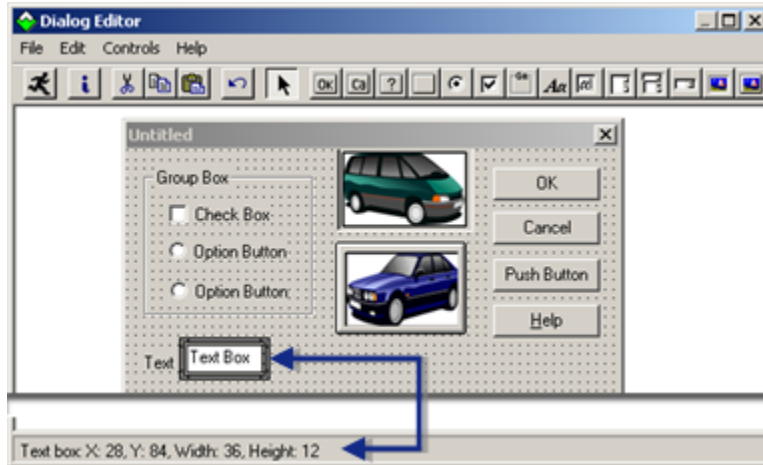
3. Click the mouse button.

Results

- The selected control is placed in the dialog box. The upper left corner of the control corresponds to the position of the pointer's crossbar at the moment you clicked the mouse button.
- The object's upper left corner coordinates, width and height display on the Dialog Editor status bar.

Note: The status bar displays this information anytime the mouse passes over a control or the control is selected.

- A frame that surrounds the object identifies it as selected.

**Tip:**

Press Ctrl+D on the keyboard to add another control that is the same type as the control that was just added.

2.3. Use the Grid to Position Controls within a Dialog Box

**Note:**

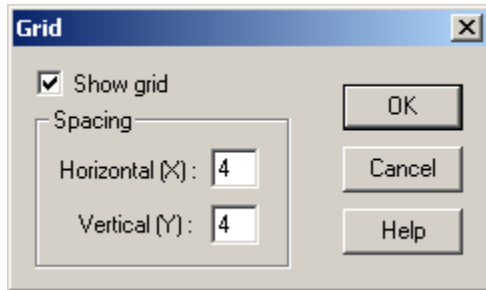
Dialog units represent increments of the font (8 point Helvetica) in which the Dialog Editor creates dialog boxes.

Unit	Represents an increment equal to:
X	1/4 of the font.
Y	1/8 of the font

1. Do one of the following.
 - Click Edit>Grid on the Dialog Editor menu bar.
 - Press Ctrl+G on the keyboard.

A Grid dialog box opens.

2. Specify the following.



Option	Description	
Show grid	Displays or hides a grid on the dialog box.	
	Check	Displays the grid.
	Clear	Hides the grid.
Horizontal (X)	Space between horizontal grid marks. The higher the number the further apart the grid marks.	
Vertical (Y)	Space between vertical grid marks. The higher the number the further apart the grid marks.	



Important:

The X and Y settings entered in the Grid dialog box remain in effect regardless of whether you choose to display the grid.

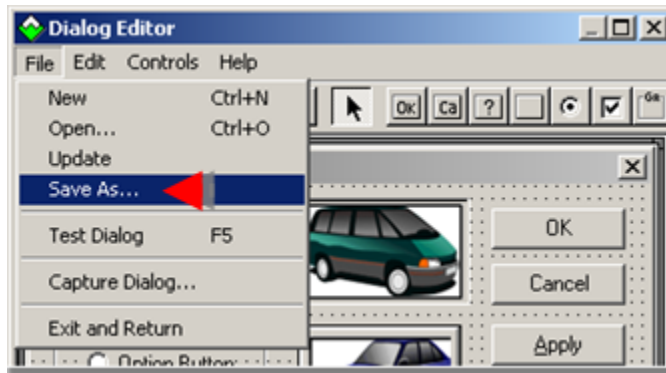
3. Click the OK button or press Enter on the keyboard.

The Dialog Editor displays or hides the grid with the settings you specified.

With the grid displayed, you can line up the crossbar on the mouse pointer with the dots on the grid to position controls precisely and align them with respect to other controls.

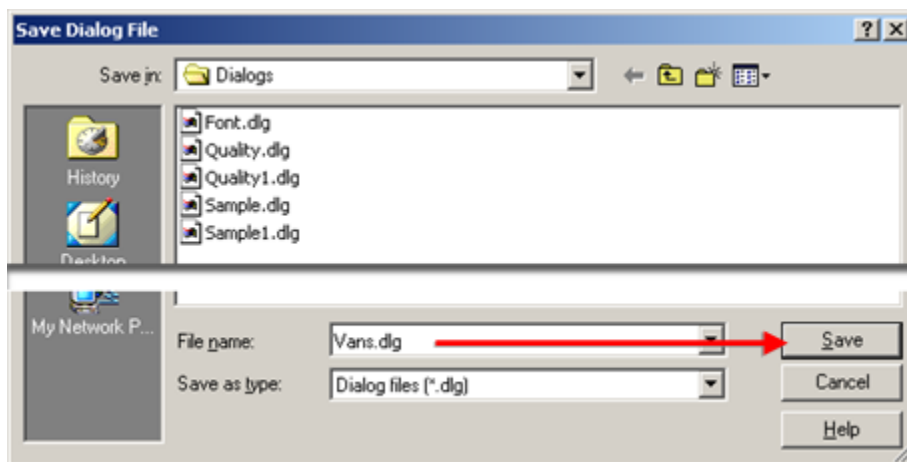
2.4. Save the Custom Dialog Box

1. Click File>Save As on the Dialog Editor menu bar.



A Save As dialog box opens.

2. Enter a name in the File name field.
3. Click Save.



The dialog box is saved as a .dlg file. The .dlg file can be moved, opened, modified, saved with a different name at any time.

3. Edit a Custom Dialog Box

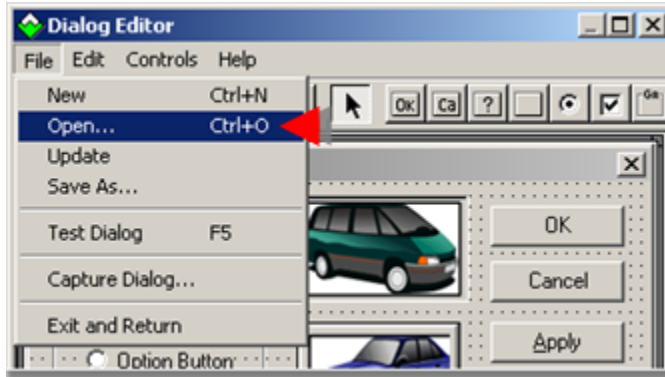
3. Edit a Custom Dialog Box

In the preceding section, you learned how to create controls and determine where they initially appear within your dialog box. In this section, you'll learn how to make various types of changes to both the dialog box and the controls in it. The following topics are included:

3.1 (<i>on page 165</i>)	Open a dialog box template.
3.2 (<i>on page 167</i>)	Select items.
3.3 (<i>on page 168</i>)	Specify tabbing order.
3.4 (<i>on page 169</i>)	Use the Information dialog box.
3.5 (<i>on page 184</i>)	Change the Position of an Item
3.6 (<i>on page 185</i>)	Change the size of an item.
3.7 (<i>on page 187</i>)	Change titles and labels.
3.8 (<i>on page 188</i>)	Assign accelerator keys.
3.9 (<i>on page 190</i>)	Duplicate and delete controls.

3.1. Open a Dialog Box Template

1. Do one of the following.
 - Click File>Open on the Dialog Editor menu bar.
 - Press Ctrl+O on the keyboard.



A message opens asking if you want to save the template for the current dialog box.

Do one of the following.

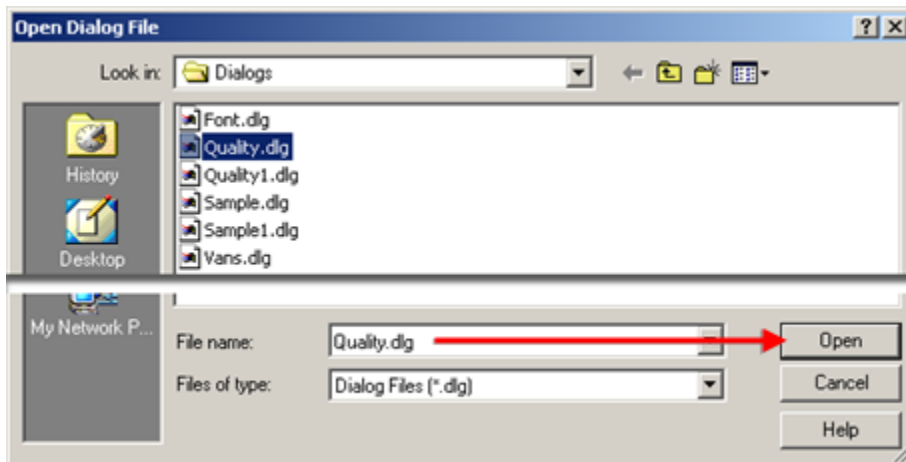
- Click Yes.

The current dialog box template code is [inserted \(on page 191\)](#) into the Program Editor script.

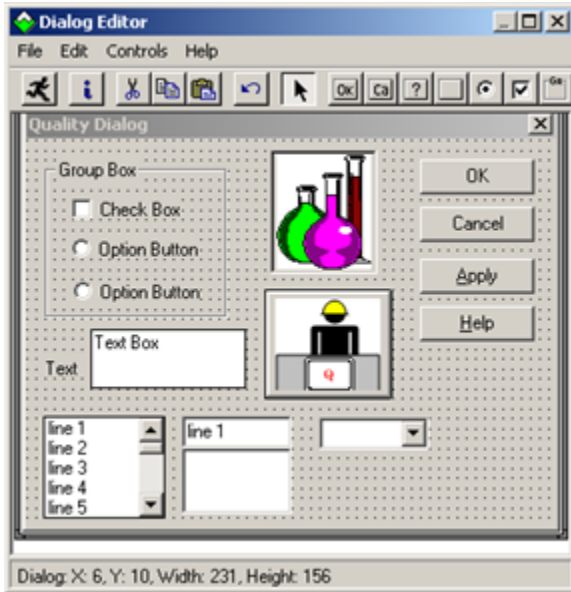
- Click No.

The Open Dialog File dialog box opens after either selection.

2. Select the file containing the dialog box template that you want to edit.
3. Click Open.



The Dialog Editor creates a dialog box from the statements in the template and displays it in the application window.

**Note:**

If there are any errors in the statements that describe the dialog box, the Dialog Translation Errors dialog box will open when you attempt to load the file into Dialog Editor. This dialog box shows the lines of code containing the errors and provides a brief description of the nature of each error.

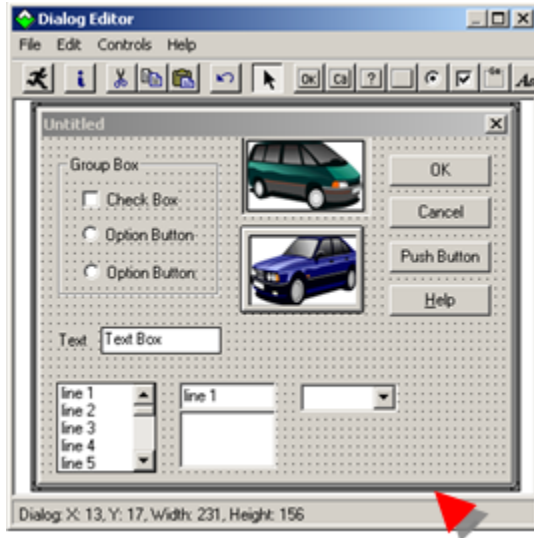
3.2. Select Items

- Select the dialog box.
- Select a control.

Select the Dialog Box

1. Click the [Select \(on page 151\)](#) button on the Dialog Editor toolbar.
2. Do one of the following.
 - Click the cursor on the title bar of the dialog box or on an empty area within the borders of the dialog box
 - Press the Tab key repeatedly until the focus moves to the dialog box.

Result: The selected dialog box is framed by a border.



Control

3. Click the [Select \(on page 151\)](#) button on the Dialog Editor toolbar.
4. Do one of the following.
 - Click the cursor on the control to be selected.
 - Press the Tab key repeatedly until the focus moves to the control.

The selected control is framed by a border.



3.3 Specify Tabbing Order

- Default tabbing order.
- Edit tabbing order.

Default Tabbing Order

The Dialog Editor creates the tabbing order based on the order in which you create the controls, not the position of the controls in the dialog box.

The closer you can come to creating controls in the order in which you want them to receive the tabbing focus, the fewer tabbing-order adjustments you will have to make later on.

Edit Tabbing Order

When a control is pasted into the dialog box, the Dialog Editor places the descriptions of at the end of the dialog box template.

Therefore, you can use a simple cut-and-paste technique to adjust the tabbing order.

1. Determine what item should be the starting point in the dialog box.

Note: Because the OK button and Cancel button display on a new dialog box by default, the OK button is the starting point and the Cancel button is the first tab.

2. [Cut \(on page 151\)](#) and [paste \(on page 151\)](#) the controls to establish the desired tabbing order.

Note: If the controls were place in the dialog box in the correct order you can simply cut/paste the OK and Cancel buttons so they are tabbed to last.

3. [Test \(on page 204\)](#) the tabbing order to make sure it is correct. Items that users cannot interact with, e.g. group boxes, are not included in the tabbing order.

3.4. Use Information Dialog Boxes

3.4. Use Information Dialog Boxes

Information dialog box enable you to check and adjust various attributes of dialog boxes and dialog box items.

3.4.1 <i>(on page 170)</i>	Dialog box information
-------------------------------	------------------------

3.4.2 (on page 171)	Control information
---------------------------	---------------------

3.4.1. Dialog Box Information

- Open the Dialog Box Information dialog box.
- Dialog Box Information dialog box options.

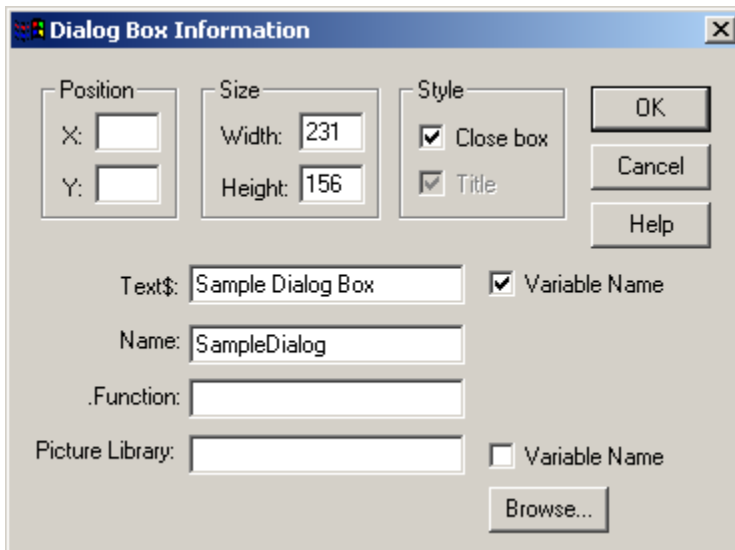
Open the Dialog Box Information dialog box.

1. Select (on page 167) the dialog box.
2. Do one of the following.
 - Click the [Information \(on page 150\)](#) button on the Dialog Editor toolbar.
 - Click Edit>Info on the Dialog Editor menu bar.
 - Double-click an empty space in the dialog box.


The Dialog Box Information dialog box opens when you use any method.

Dialog Box Information dialog box options

Options for the dialog box are as follows.



At-tribute	Description

Position	(Optional) Dialog box position (dialog units (on page 162)) in the window/screen in which it opens	
	Coordinate	Units from the:
	X	Left side of the window/screen.
	Y	Top of the window/screen.
Size	Dialog box size includes the number of dialog units (on page 162) in the:	
	Width	Dialog box width
	Height	Dialog box height.
Style	(Optional) Check to display the following.	
	Close box	Close box button 
	Title	Dialog box title bar. Note: Title is enabled if Close box is cleared. If Close box is checked so the Close box button displays, the title bar must display.
Text\$	(Optional) Text displayed on the title bar of the dialog box.	
	Variable Name	Check to identify the Text\$ entry as a variable. Note: if Text\$ is a variable, spaces cannot be used in the entry.
Name	Name used for the dialog box template in script code	
.Function	(Optional) Name of a script function in your dialog box	
Picture Library	(Optional) Picture library (.dll file) from which one or more pictures in the dialog box are obtained	
	Variable Name	Check to identify the Picture Library as a variable name.
	Browse	Opens a Select a Picture Library browser to help find the .dll file.

3.4.2. Control Information

3.4.2. Control Information

1. [Select \(on page 168\)](#) an item in the dialog box.
2. Do one of the following.
 - Click the [Information \(on page 150\)](#) button on the Dialog Editor toolbar.
 - Click Edit>Info on the Dialog Editor menu bar.
 - Double-click the control.

The <Item> Information dialog box opens when you use any method.

3. Configure the <Item> Information dialog box.

3.4.2.1 (on page 173)	Check Box Information
3.4.2.2 (on page 174)	Combo Box Information
3.4.2.3 (on page 175)	Drop List Box Information
3.4.2.4 (on page 175)	Group Box Information
3.4.2.5 (on page 176)	List Box Information
3.4.2.6 (on page 177)	Picture Information
3.4.2.7 (on page 179)	Picture Button Information
3.4.2.8 (on page 180)	Push Button Information

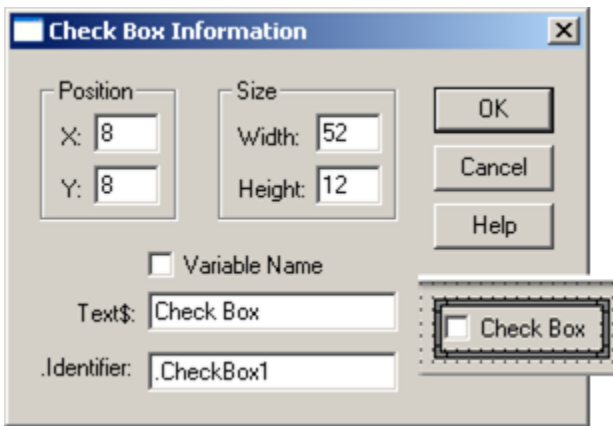
3.4.2.9 <i>(on page 181)</i>	Option button Information
3.4.2.10 <i>(on page 181)</i>	Text Information
3.4.2.11 <i>(on page 182)</i>	Text Box Information

4. Do one of the following.

- Click OK to save the configuration.
- Click Cancel to discard the changes and close the Information dialog box.

3.4.2.1. Check Box Information

Options in the Check Box Information dialog box are as follows.

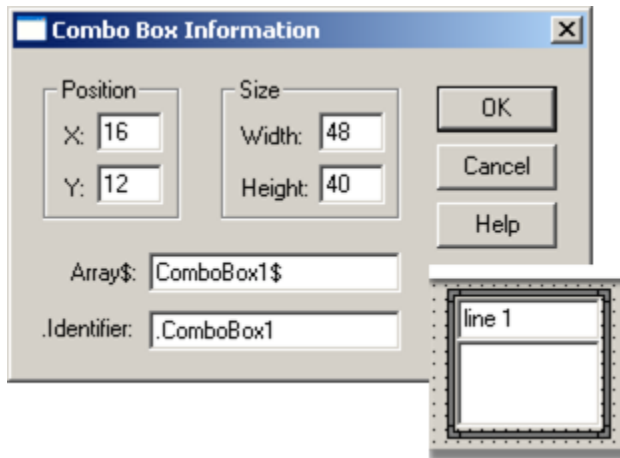


Option	Description	
Position	Position of the check box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the checkbox, including the label, in dialog units (on page 162) .	
	Width	Width of the checkbox and label

	Height	Height of the checkbox and label.
Variable Name	Check to identify the Text\$ entry as a variable.	
Text\$	(Optional) Text that displays as the checkbox label. Note: if Text\$ is a variable, spaces cannot be used in the entry.	
.Identifier	(Optional) Name used for the checkbox in a script's code.	

3.4.2.2. Combo Box Information

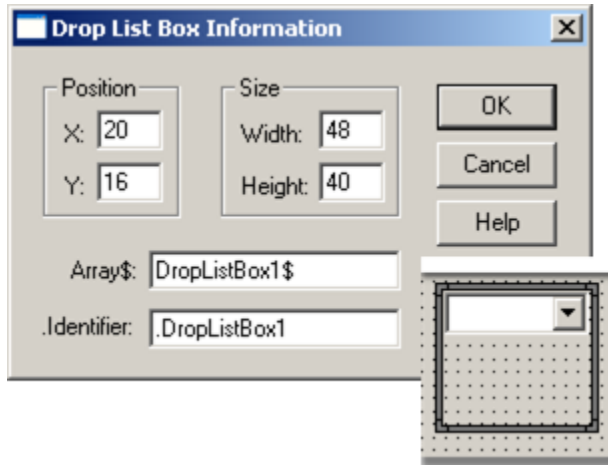
Options in the Combo Box Information dialog box are as follows.



Option	Description	
Position	Position of the combo box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the combo box in dialog units (on page 162) .	
	Width	Width of the combo box.
	Height	Height of the combo box.
Array\$	Name of the array variable in a script's code.	
.Identifier	(Optional) Name used for the combo box in a script's code.	

3.4.2.3. Drop List Box Information

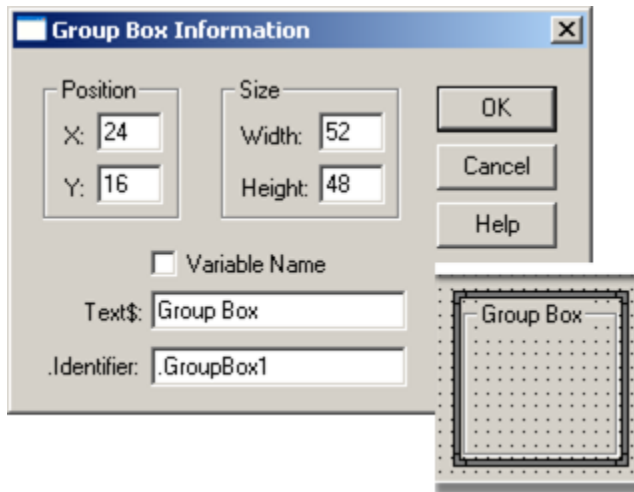
Options in the List Box Information dialog box are as follows.



Option	Description	
Position	Position of the drop list box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the drop list box in dialog units (on page 162) .	
	Width	Width of the list box.
	Height	Height of the list box.
Array\$	Name of the array variable in a script's code.	
.Identifier	(Optional) Name used for the drop list box in a script's code.	

3.4.2.4. Group Box Information

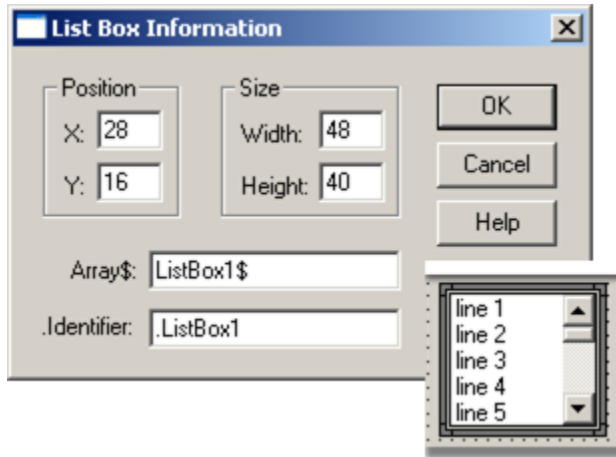
Options in the Group Box Information dialog box are as follows.



Option	Description	
Position	Position of the group box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the group box in dialog units (on page 162) .	
	Width	Width of the list box.
	Height	Height of the list box.
Variable Name	Check to identify the Text\$ entry as a variable.	
Text\$	(Optional) Text that displays as the group box label. Note: if Text\$ is a variable, spaces cannot be used in the entry.	
.Identifier	(Optional) Name used for the group box in a script's code.	

3.4.2.5. List Box Information

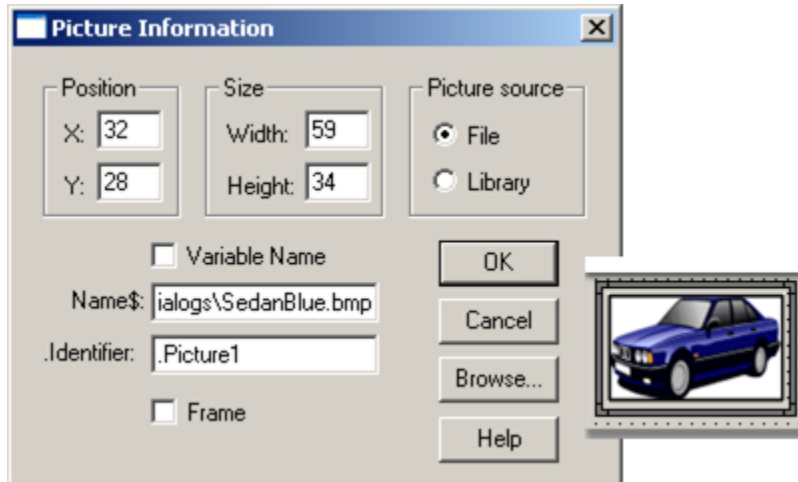
Options in the List Box Information dialog box are as follows.



Option	Description	
Position	Position of the list box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the list box in dialog units (on page 162) .	
	Width	Width of the list box.
	Height	Height of the list box.
Array\$	Name of the array variable in a script's code.	
.Identifier	(Optional) Name used for the list box in a script's code.	

3.4.2.6. Picture Information

Options in the Picture Information dialog box are as follows.

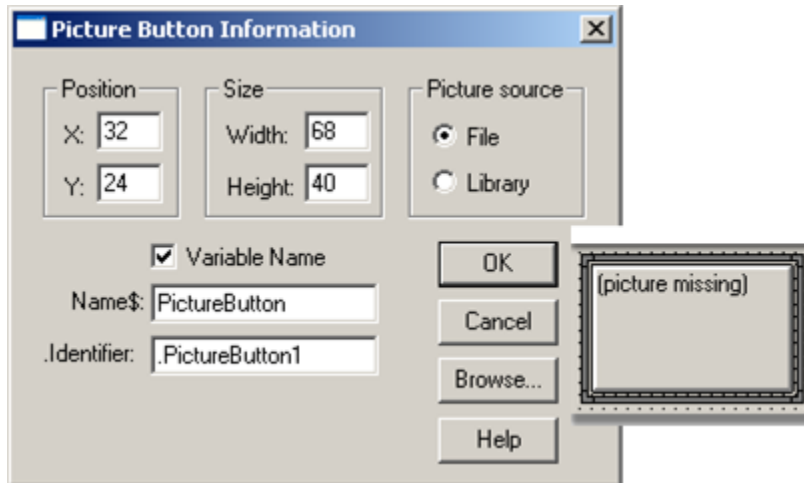


Option	Description	
Position	Position of the picture in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the picture in dialog units (on page 162) .	
	Width	Width of the picture.
	Height	Height of the picture.
Variable Name	Check to identify the Name\$ entry as a variable.	
Picture source	Picture source selections are:	
	File	A .bmp or .wmf file.
	Library	Included in a library .dll file.
Name\$	Either an absolute or variable name can be entered.	
	Absolute	Path and name of the picture button file.
	Variable	Name with no spaces or wild card characters. <ul style="list-style-type: none"> • An underscore (_) can be included in the name. • The picture will be identified as missing in the dialog box template.

.Identifier	(Optional) Name used for the picture in a script's code.
-------------	--

3.4.2.7. Picture Button Information

Options in the Picture Button Information dialog box are as follows.

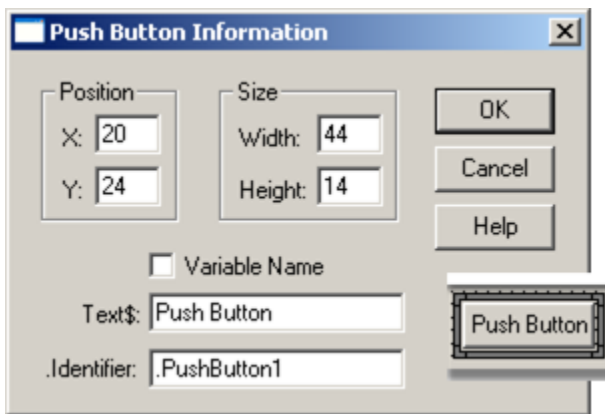


Option	Description	
Position	Position of the picture button in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the picture button in dialog units (on page 162) .	
	Width	Width of the picture button.
	Height	Height of the picture button.
Picture source	Picture button source selections are:	
	File	A .bmp or .wmf file.
	Library	Included in a library .dll file.
Variable Name	Check to identify the Name\$ entry as a variable.	
Name\$	Either an absolute or variable name can be entered.	
	Absolute	Path and name of the picture button file.

	Variable	Name with no spaces or wild card characters. <ul style="list-style-type: none"> • An underscore (_) can be included in the name. • The picture will be identified as missing in the dialog box template.
.Identifier	(Optional) Name used for the picture in a script's code.	

3.4.2.8. Push Button Information

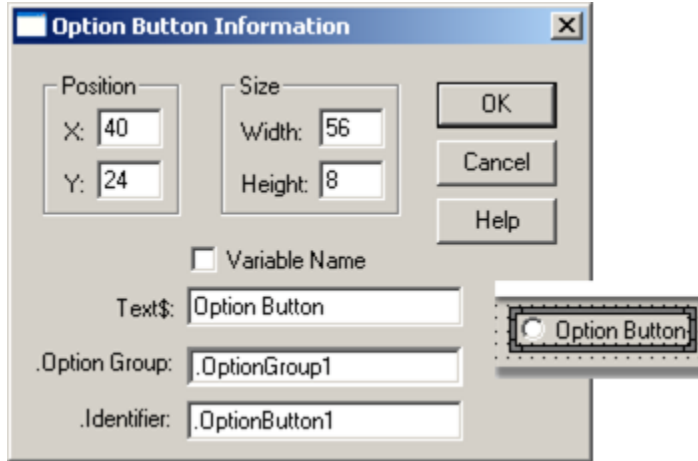
Options in the Push Button Information dialog box are as follows.



Option	Description	
Position	Position of the Push Button box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the push button in dialog units (on page 162) .	
	Width	Width of the push button.
	Height	Height of the push button
Variable Name	Check to identify the Text\$ entry as a variable.	
Text\$	(Optional) Text that displays as the push button label. Note: if Text\$ is a variable, spaces cannot be used in the entry.	
.Identifier	(Optional) Name used for the push button in a script's code.	

3.4.2.9. Option Button Information

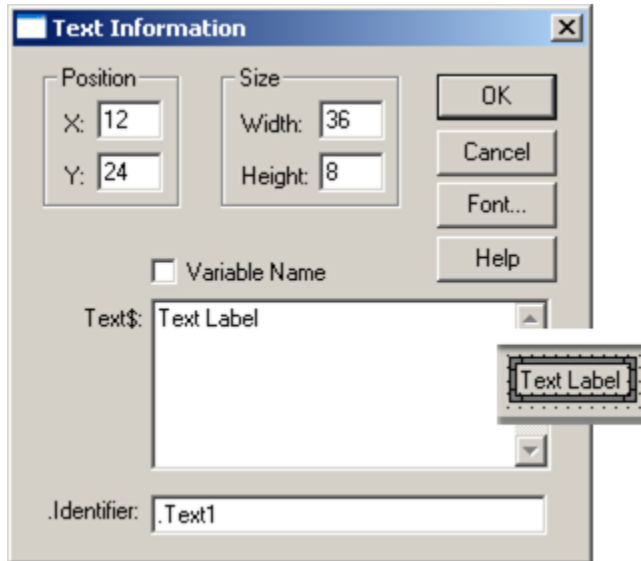
Options in the Option Button Information dialog box are as follows.



Option	Description
Position	Position of the option button in the dialog box from the:
X	Left of the dialog box.
Y	Top of the dialog box.
Size	Size of the option button, including the label, in dialog units (on page 162) .
Width	Width of the option button and label
Height	Height of the option button and label.
Variable Name	Check to identify the Text\$ entry as a variable.
Text\$	(Optional) Text that displays as the checkbox label. Note: if Text\$ is a variable, spaces cannot be used in the entry.
.Option Group	Name referring to a group of option buttons in a script's code.
.Identifier	(Optional) Name used for the checkbox in a script's code.

3.4.2.10. Text Information

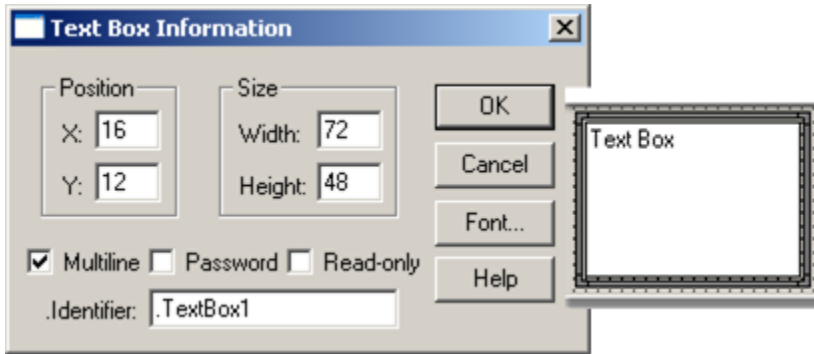
Options in the Text Information dialog box are as follows.



Option	Description	
Position	Position of the text in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the text in dialog units (on page 162) .	
	Width	Width of the text.
	Height	Height of the text.
Variable Name	Check to identify the Text\$ entry as a variable.	
Text\$	Text that displays up to 255 characters Note: if Text\$ is a variable, spaces cannot be used in the entry.	
Identifier	(Optional) Name used for the text in a script's code.	
Font	Opens a Font dialog box to select the text font type, style and size.	

3.4.2.11. Text Box Information

Options in the Text Box Information dialog box are as follows.



Option	Description	
Position	Position of the text box in the dialog box from the:	
	X	Left of the dialog box.
	Y	Top of the dialog box.
Size	Size of the text in dialog units (on page 162) .	
	Width	Width of the text box.
	Height	Height of the text box.
Multiline	Do one of the following.	
	Check	Enable text wrapping.
	Clear	Disable text wrapping.
Pass- word	Do one of the following.	
	Check	Require a password for a user to make a text entry if the text box is read/write.
	Clear	Allow text entries without a password if the text box is read/write.
Read- only	Do one of the following.	
	Check	The text box is read-only.
	Clear	The text box is read/write.
Text\$	Text that displays up to 255 characters Note: if Text\$ is a variable, spaces cannot be used in the entry.	

Identifier	(Optional) Name used for the text in a script's code.
Font	Opens a Font dialog box to select the text font type, style and size.

3.5. Change the Position of an Item

The Dialog Editor provides several ways to reposition dialog boxes and items.

- Mouse.
- Arrow keys.
- Dialog Box Information dialog box.
- Item with the Information dialog box.

Mouse

1. Click the [Select \(on page 151\)](#) button on the Dialog Editor toolbar.
2. Place the cursor on an empty area in the dialog box or on a control.
3. Hold the left-mouse button down and drag the dialog box or control to the desired location.



Note:

The increments by which you can move a control with the mouse are governed by the [grid setting \(on page 162\)](#).

Example

The grid has the following settings

- X = 4
- Y = 6

The control can move in the following increments:

- Horizontal = 4 X units
- Vertical = 6 Y units.

This feature is useful if you are trying to align controls in your dialog box. If you want to move controls in smaller or larger increments.

Arrow Keys

4. Select the dialog box or control that will be moved.

5. Do one of the following.

- Press an Arrow key once to move the item by 1 X or 1 Y unit in the desired direction.
- Hold the arrow key down to move the item steadily (in increments of 1 unite) along in the desired direction.



Note:

When you reposition an item with the arrow keys, a faint, partial afterimage of the item may remain visible in the item's original position. These afterimages are rare and will disappear once you test your dialog box.

Dialog Box with the Dialog Box Information Dialog Box

6. [Open \(on page 170\)](#) the Dialog Box Information dialog box.
7. Change the X and Y coordinates in the **Position** group box.
8. Click the OK button or press Enter on the keyboard.

If you specified X and Y coordinates, the dialog box moves to that position. If you left the X coordinate blank, the dialog box will be centered horizontally static to the parent window of the dialog box when the dialog box is run. If you left the Y coordinate blank, the dialog box will be centered vertically static to the parent window of the dialog box when the dialog box is run.

Item with the Information Dialog Box

9. [Open \(on page 170\)](#) the Information dialog box for the control that you want to move.
10. Change the X and Y coordinates in the **Position** group box.
11. Click the OK button or press Enter on the keyboard.



Note:

When you move a dialog box or control with the arrow keys or with the Information dialog box, the item's movement is not restricted to the increments specified in the grid setting. When you attempt to test a dialog box containing hidden controls (i.e., controls positioned entirely outside the current borders of your dialog box), the Dialog Editor displays a message advising you that there are controls outside the dialog box's borders and asks requests confirmation to proceed with the test. If you proceed, the hidden controls will be disabled for testing purposes.

3.6. Change the Size of an Item

Dialog boxes and controls can be resized either by directly manipulating them with the mouse or by using the Information dialog box. Certain controls can also be resized automatically to fit the text displayed on them.

- Resize an item with the mouse.
- Resize an item with the Information dialog box.
- Resize selected controls automatically.

Resize an Item with the Mouse

1. Click the [Select \(on page 151\)](#) button on the Dialog Editor toolbar.
2. Place the cursor over a border or corner of the dialog box or a control..
3. Hold the the left-mouse button down and drag the border or corner until the dialog box or control reaches the desired size.

Resize an Item with the Information Dialog Box

4. [Open \(on page 169\)](#) the Information dialog box for the dialog box or a selected control.
5. Change the **Width** and **Height** settings in the **Size** group box.
6. Click OK or press Enter on the keyboard.

Resize Selected Items Automatically

The following controls can be resized automatically.

- Option button
 - Text control
 - Push button
 - Check box
 - Text box
7. Click the [Select \(on page 151\)](#) button on the Dialog Editor toolbar.
 8. Select the control to be resized.
 9. Press F2 on the keyboard..

The borders of the control will expand or contract to fit the text displayed on it.



Note:

Picture controls and picture button controls must be resized manually.

- Windows metafiles always expand or contract proportionally to fit within the picture control or picture button control containing them.
- Windows bitmaps are of a fixed size.

If you place a bitmap in a control that is:

- Smaller than the bitmap, the bitmap is clipped off on the right and bottom.
- Larger than the bitmap, the bitmap is centered within the borders of the control.

3.7. Change Titles and Labels

- Default titles and labels.
- Change a title of label.

Default titles and labels

- The default titles for a dialog box is Untitled.
- Default labels for the following controls and items are generic, as follows.

Item	Default Title or Label
Dialog box	Untitled
Check boxes	Check Box
Group boxes	Group Box
Option buttons	Option Button
Push buttons	Push Button
Text	Text



Note:

- The OK and Cancel buttons also have labels that cannot be changed.
- The following controls do not have their own labels. You can position a text control above or beside these controls to serve as a de facto label
 - Combo boxes
 - Drop list boxes

- List boxes
- Pictures
- Picture buttons.
- Text boxes

Change a title or label

1. [Open \(on page 169\)](#) the Information dialog box for the dialog box title you want to change or for the control label you want to change.
2. Enter the new title or label in the **Text\$** field.

Note: Dialog box titles and control labels are optional; you can leave the **Text\$** field blank.

3. Check the Variable Name checkbox if the information in the **Text\$** field should be interpreted as a variable name rather than a literal string.
4. Click the OK button or press Enter on the keyboard.

The Information dialog box closes; the new title or label displays in the title bar or on the control.

3.8. Assign Accelerator Keys

- Accelerator key definition.
- Create an accelerator key.
- Guidelines for accelerator keys.

Accelerator key definition

Accelerator keys:

- Enable users to access dialog box controls by pressing Alt + <specified letter>.
- Are essentially a single letter from a control's label.

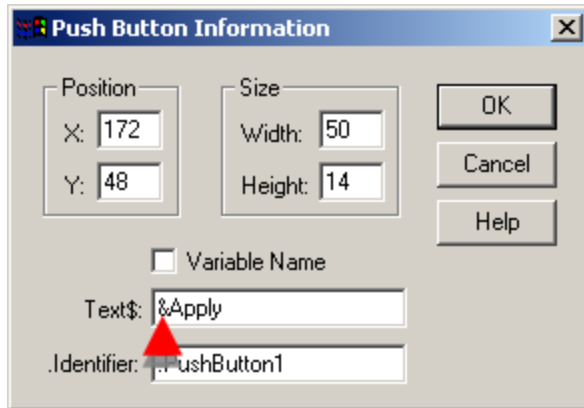
Users can employ accelerator keys to:

- Choose a push button or an option button.
- Toggle a check box on or off and move the insertion point into one of the following.
 - Text box
 - Group box
 - Currently selected item in one of the following.
 - List box.

- Combo box.
- Drop list box..

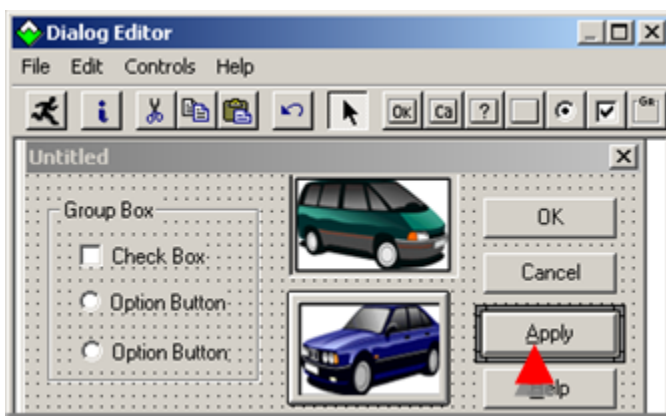
Create an accelerator key

1. Open the Information dialog box for the control that will have an assigned an accelerator key.
2. Select the **Text\$** field.
3. Type an ampersand (&) before the letter designated as the accelerator key.



4. Click the OK button or press Enter on the keyboard.

The designated letter is now underlined on the control's label; users will be able to access the control by pressing Alt +<underlined letter>.



Guidelines for accelerator keys

- Accelerator keys can be assigned directly to the following controls that have their own label
- Check boxes
- Group boxes

- Option buttons
- Push buttons
- The OK and Cancel buttons cannot have accelerator keys

Note: OK and Cancel labels cannot be edited.

- A de facto accelerator key can be created for certain controls that do not have their own labels by assigning an accelerator key to an associated text control.
- Combo boxes
- Drop list boxes
- List boxes
- Text boxes
- Accelerator key assignments must be unique within a particular dialog box. If you attempt to assign the same accelerator key to more than one control, a message displays reporting that the letter has already been assigned.



Note:

In order for such a de facto accelerator key to work properly, the text control or group box label to which you assign the accelerator key must be associated with the control(s) to which you want to provide user access that is, in the dialog box template, the description of the text control or group box must immediately precede the description of the control(s) that you want associated with it. The simplest way to establish such an association is to create the text control or group box first, followed immediately by the associated control(s)

3.9. Duplicate and Delete Controls

- Duplicate a Control.
- Delete a Single Control.
- Delete all the Controls in a dialog box.

Duplicate a Control

1. Select the control to duplicate.
2. Do one of the following.
 - Click Edit>Duplicate on the Dialog Editor menu bar.
 - Press Ctrl+D on the keyboard.

A duplicate copy of the selected control displays in your dialog box.

3. Repeat **2** as many times as necessary to create the desired number of duplicate controls.

Result: The selected control is duplicated each time duplication is repeated.

Delete a Single Control

4. Select the control to delete.
5. Press Delete on the keyboard.

Result: The selected control is deleted from the dialog box.

Delete all the Controls in a Dialog Box

6. Select the dialog box.
7. Press Delete on the keyboard.

If the dialog box contains more than one control a message displays asking:

Do you want to delete all controls from the dialog box?

8. Click the Yes button or press Enter on the keyboard.

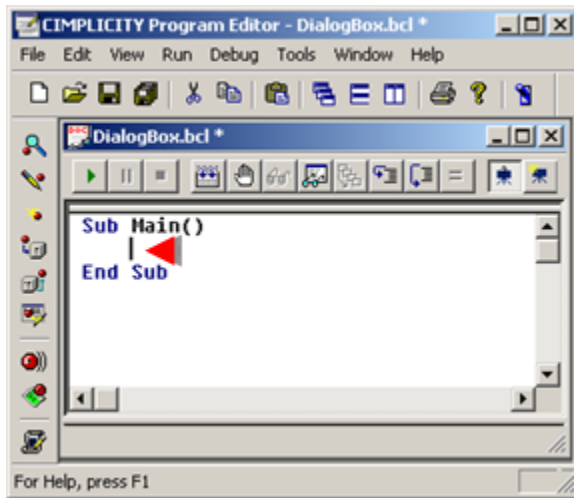
All the controls are deleted, but the dialog box's title bar and close box (if displayed) remain unchanged.

4. Insert/Paste a Dialog Box Template Code into a Script

- Insert template code a Program Editor script
- Paste template code into a Program Editor script.

Insert Template Code into a Program Editor Script

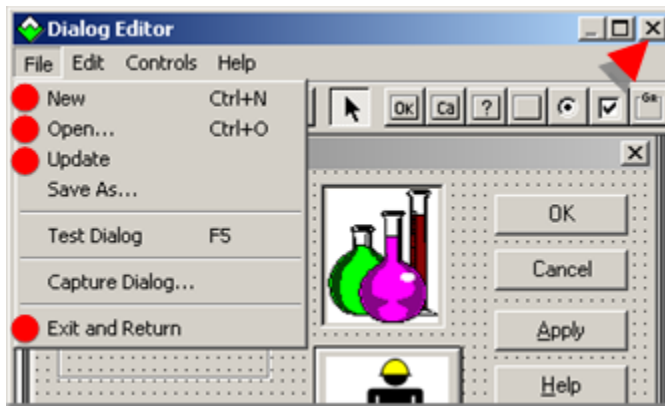
1. Place the cursor in a Program Editor script where the dialog box code will be inserted.



2. Click Edit>Insert Picture on the Program Editor menu bar.

The Dialog Editor opens displaying a new dialog box.

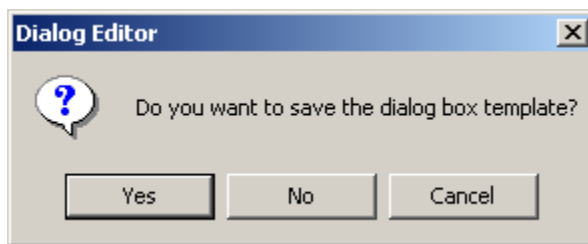
3. Do one of the following.
 - Configure the new dialog box.
 - Click File>Open on the Dialog Editor menu bar to open an existing dialog box.
4. Do one of the following.



Click	On the
File>New	Menu bar
File>Open	Menu bar
File>Update	Menu bar

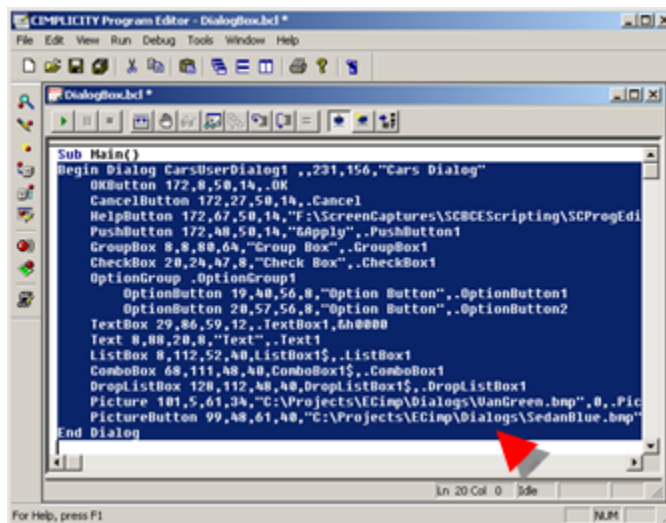
File>Exit and Return	Menu bar
Close button	Title bar
Press	On the
Ctrl+N	Key-board
Ctrl+O	Key-board

A message box opens asking if you want to save the dialog box template.



5. Click Yes.

Result: The dialog box template code is inserted into the Program Editor script at the specified location.

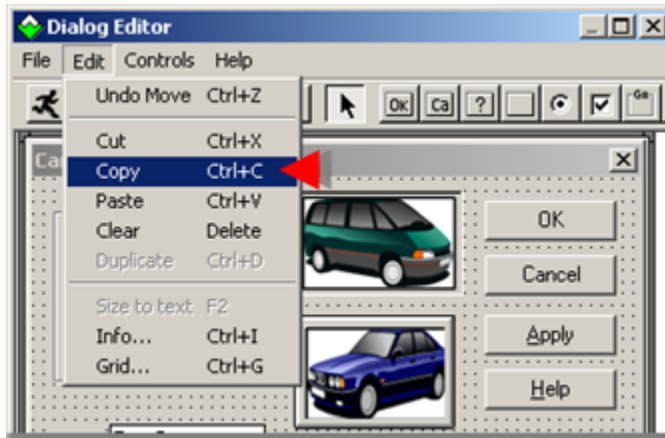


Paste Template Code into the Program Editor Script

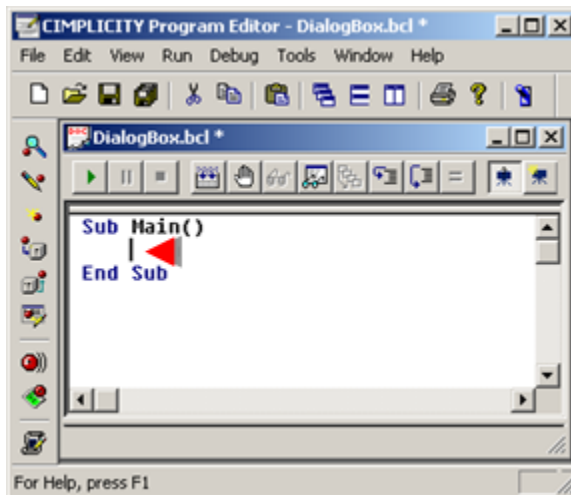
6. Click Edit>Insert Picture on the Program Editor menu bar.

The Dialog Editor opens displaying a new dialog box.

7. Do one of the following.
 - Configure the new dialog box.
 - Click File>Open on the Dialog Editor menu bar to open an existing dialog box.
8. [Select \(on page 167\)](#) the dialog box.
9. Do one of the following.
 - Click Edit>Copy on the Dialog Editor menu bar.
 - Press Ctrl+C on the keyboard.

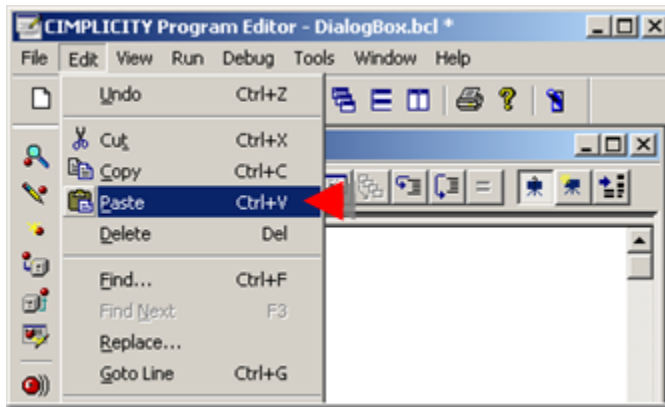


10. Select the Program Editor.
11. Place the cursor where the code should be inserted.

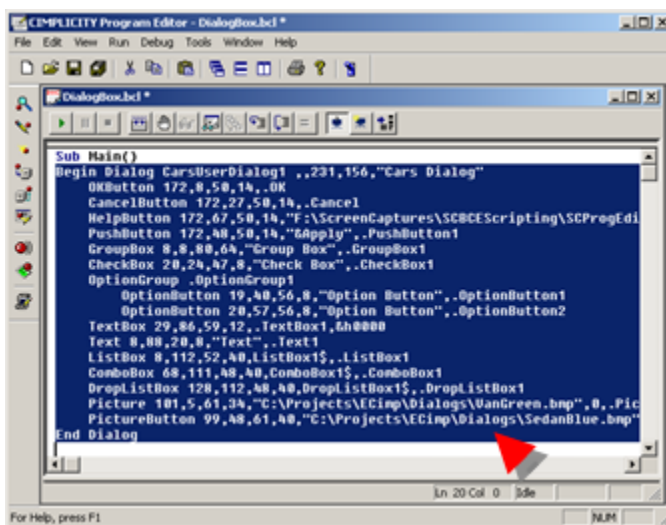


12. Do one of the following.

- Click Edit>Paste on the Program Editor menu bar.
- Press Ctrl+P on the keyboard.



The Dialog script is pasted at the insertion point in the Program Editor script.



5. Edit an Existing Dialog Box

5. Edit an Existing Dialog Box

There are three ways to edit an existing dialog box: (1) You can copy the template of the dialog box you want to edit from a script to the Clipboard and paste it into Dialog Editor. (2) You can use the capture feature to "grab" an existing dialog box from another application and insert a copy of it into Dialog Editor. (3) You can open a dialog box template file that has been saved on a disk.

5.1 (on page 196)	Paste an existing dialog box into the Dialog Editor.
5.2 (on page 199)	Capture a dialog box from another application.

5.1. Paste Program Editor Code into the Dialog Editor

You can use the Dialog Editor to modify the statements in your script that correspond to an entire dialog box or to one or more dialog box controls.

Paste the following into the Dialog Editor.

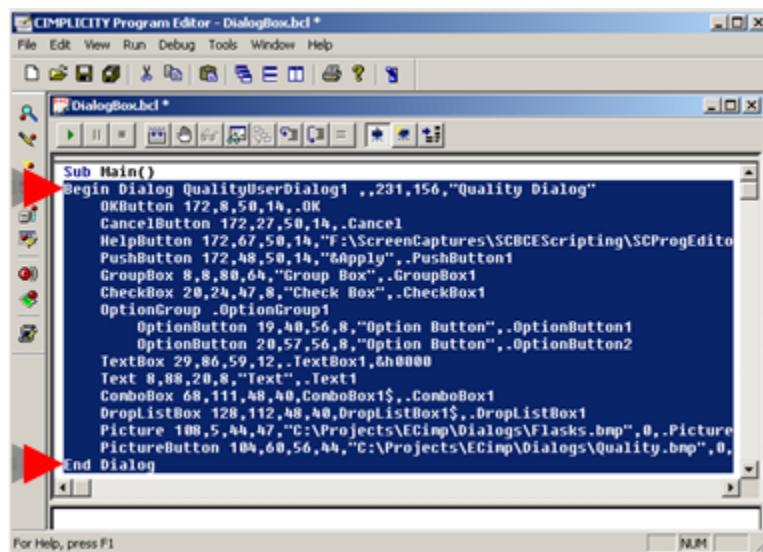
- An existing dialog box.
- One of more controls from an existing dialog box.
- Notes for pasting code into a dialog box.

Paste Dialog Box Code into a Dialog box Template

1. Select code in the Program Editor for the entire dialog box.

The code:

- Begins with [Begin Dialog \(on page 348\)](#) (statement).
- Ends with [End Dialog \(on page 487\)](#) (statement).

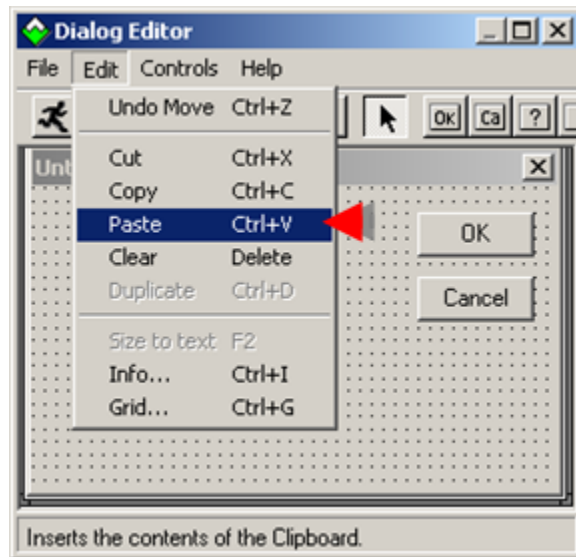


```

Sub Main()
Begin Dialog QualityUserDialog1 ,,231,156,"Quality Dialog"
  OKButton 172,8,50,14,.OK
  CancelButton 172,27,50,14,.Cancel
  HelpButton 172,67,50,14,"F:\ScreenCaptures\SCBCEScripting\SCProgEdito
  PushButton 172,48,50,14,"&Apply",.PushButton1
  GroupBox 8,8,80,64,"Group Box",.GroupBox1
  CheckBox 20,24,47,8,"Check Box",.CheckBox1
  OptionGroup .OptionGroup1
    OptionButton 19,40,56,8,"Option Button",.OptionButton1
    OptionButton 20,57,56,8,"Option Button",.OptionButton2
  TextBox 29,86,59,12,.TextBox1,&h0000
  Text 8,88,20,8,"Text",.Text1
  ComboBox 68,111,48,40,ComboBox1$,..ComboBox1
  DropListBox 128,112,48,40,DropListBox1$,..DropListBox1
  Picture 108,5,44,47,"C:\Projects\ECimp\Dialogs\Flasks.bmp",0,.Picture
  Picture 104,60,56,44,"C:\Projects\ECimp\Dialogs\Quality.bmp",0,
End Dialog

```

2. Do one of the following.
 - Click Edit>Copy on the Program Editor menu bar.
 - Press Ctrl+C on the keyboard.
3. Select the dialog box in the Dialog Editor that will be replaced.
4. Do one of the following.
 - Click Edit>Paste on the Dialog Editor menu bar.
 - Press Ctrl+V on the keyboard.

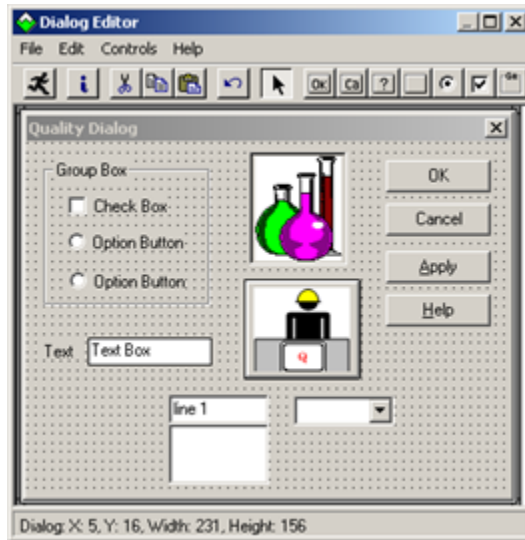


A message box opens asking:

Do you want to replace the dialog box?

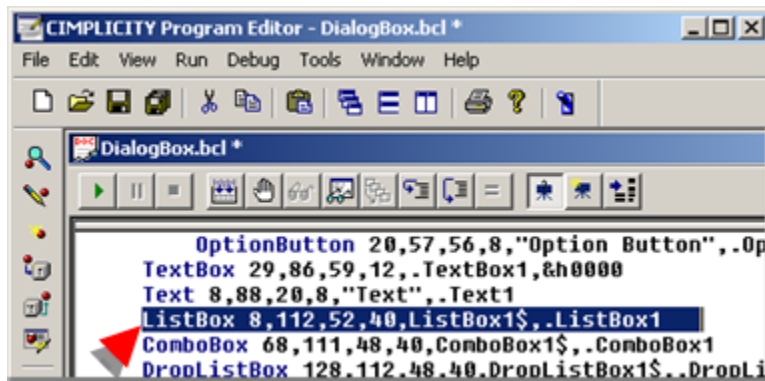
5. Click Yes.

Result: The dialog box template code copied from the Program Editor script replaces the selected dialog box template. The dialog box changes according to the pasted code.



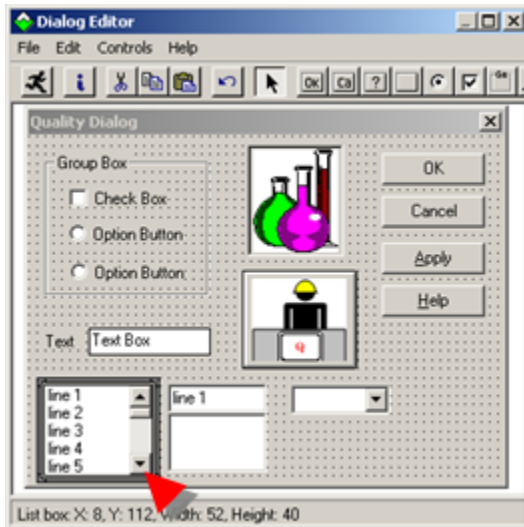
Paste Control Code into a Dialog Box Template

6. Select code in the Program Editor that defines one or more controls for a dialog box.



7. Do one of the following.
 - Click Edit>Copy on the Program Editor menu bar.
 - Press Ctrl+C on the keyboard.
8. Select the dialog box in the Dialog Editor that will be modified.
9. Do one of the following.
 - Click Edit>Paste on the Dialog Editor menu bar.
 - Press Ctrl+V on the keyboard.

The selected controls are pasted into the dialog box based on the location specifications in the code.



Notes for Pasting Code into a Dialog Box

- When you paste a dialog box template into the Dialog Editor, the tabbing order of the controls is determined by the order in which the controls are described in the template.
- When you paste one or more controls into Dialog Editor, they will come last in the tabbing order, following the controls that are already present in the current dialog box.
- If there are any errors in the statements that describe the dialog box or controls, the Dialog Translation Errors dialog box will open when you attempt to paste these statements into Dialog Editor. This dialog box shows the lines of code containing the errors and provides a brief description of the nature of each error.

5.2. Capture a Dialog Box from Another Application

The Dialog Editor provides a quick way to capture standard Windows dialog box controls from another application and insert those controls into Dialog Editor for editing.



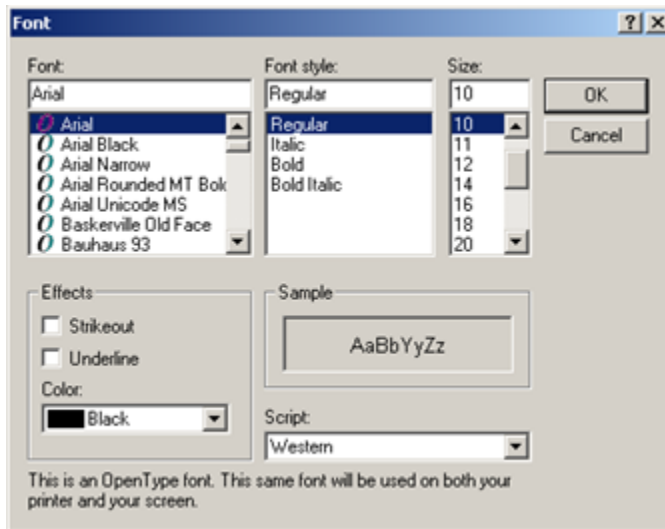
Important:

The Dialog Editor only supports standard Windows controls and standard Windows dialog boxes.

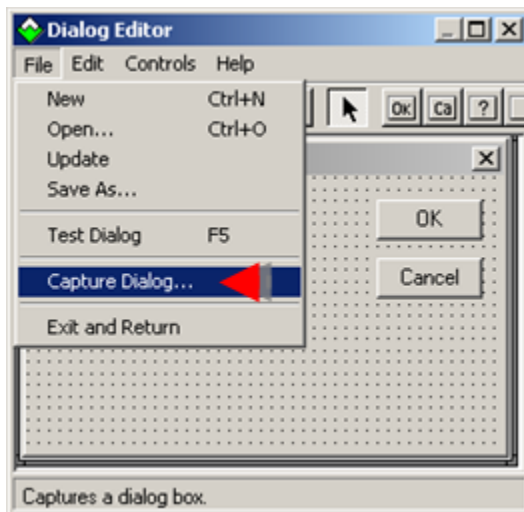
If the target dialog box

- Contains both standard Windows controls and custom controls, only the standard Windows controls will appear in Dialog Editor's application window.
- Is not a standard Windows dialog box, you will be unable to capture the dialog box or any of its controls.

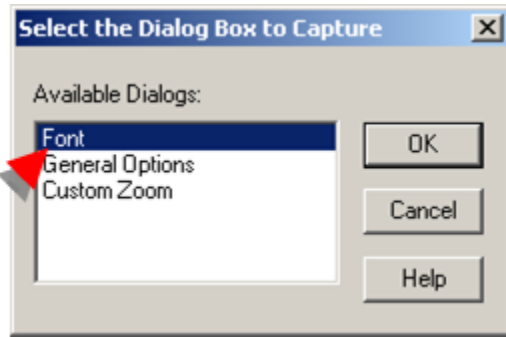
1. Open a standard Windows dialog box that has controls you want to use.



2. Open the Dialog Editor.
3. Click File>Capture Dialog on the Dialog Editor menu bar.



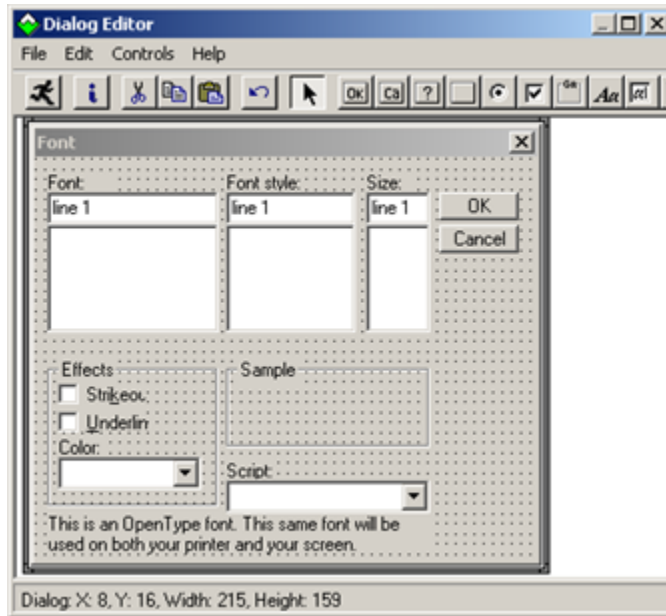
4. Select the dialog to capture.
5. Click OK.



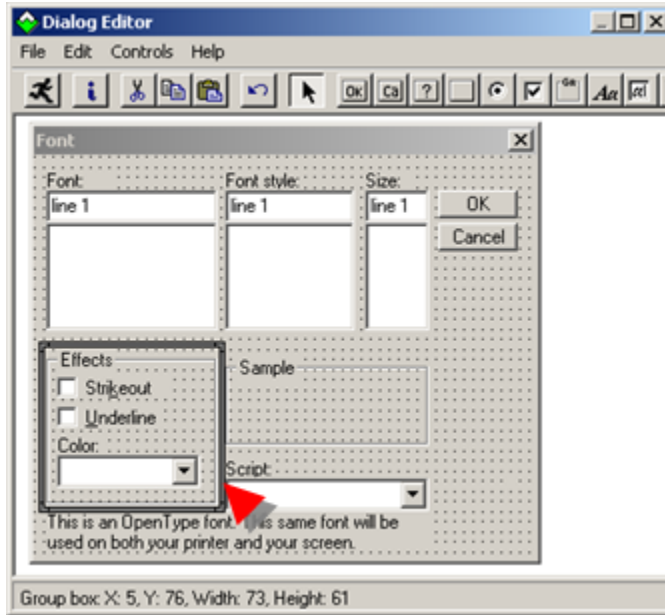
A message opens to confirm if you want to replace the dialog box that is currently in the Dialog Editor.

6. Click Yes.

The captured dialog box with standard controls replaces the current dialog box.

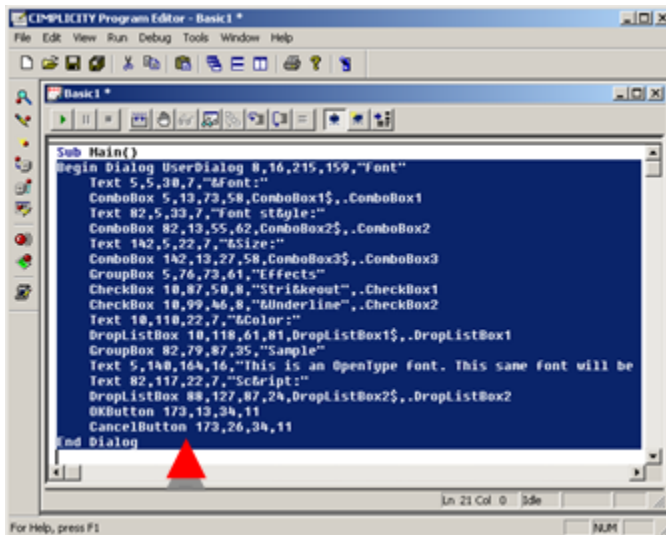


7. Modify the layout the same as you would for any other dialog box.



- Use any of the Dialog Editor methods to save the dialog box or insert/paste the dialog box template code into a script.

The captured dialog box template code displays in the Program Editor. The code can be modified to fill the script's requirements.



- Test a Dialog Box
- Test a Dialog Box

6.1 (<i>on page 203</i>)	Check for basic dialog box editing errors.
6.2 (<i>on page 204</i>)	Run the dialog box test.

6.1. Check for Basic Dialog Box Editing Errors

Following is a checklist of common errors.

It is recommended that you check the dialog box for these errors before testing it.

When all statements on the list are True, the dialog box is ready to be tested.

Statement	T/F
The dialog box contains a command button, i.e. a default OK or Cancel button, a push button, or a picture button.	
The dialog box contains all the necessary push buttons	
The dialog box contains a Help button, if required.	
All the controls are aligned correctly.	
All the controls are sized correctly.	
The font is set correctly in text controls.	
The Close button displays, if required.	
The title bar displays, if required.	
All control labels are spelled correctly.	
All control labels are capitalized correctly.	
The title is spelled correctly.	
The title is capitalized correctly.	
All the controls fit in the borders of the dialog box.	
All related controls are grouped together effectively in group boxes.	
All of the controls are labelled with their own labels or de-facto text labels.	

All of the necessary key assignments have been made.

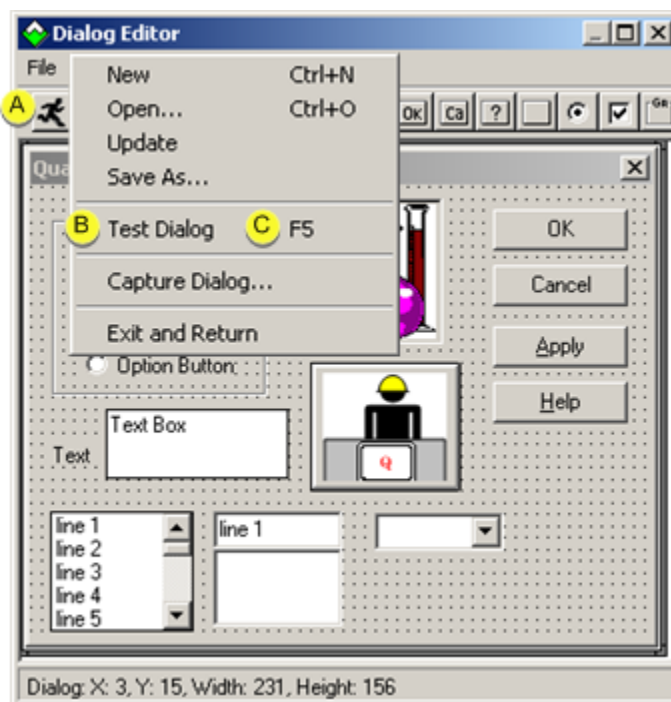
6.2. Run the Dialog Box Test

Testing a dialog box is an iterative process that involves running the dialog box to see how well it works, identifying problems, stopping the test and fixing those problems, then running the dialog box again to make sure the problems are fixed.

1 (on page 204)	Start a test.
2 (on page 205)	Test the dialog box.
3 (on page 206)	Stop the test.

Start a Test

Do one of the following.



A	Click the Test Dialog button
B	Click File>Test Dialog on the Dialog Editor menu bar.
C	Press F5 on the keyboard.

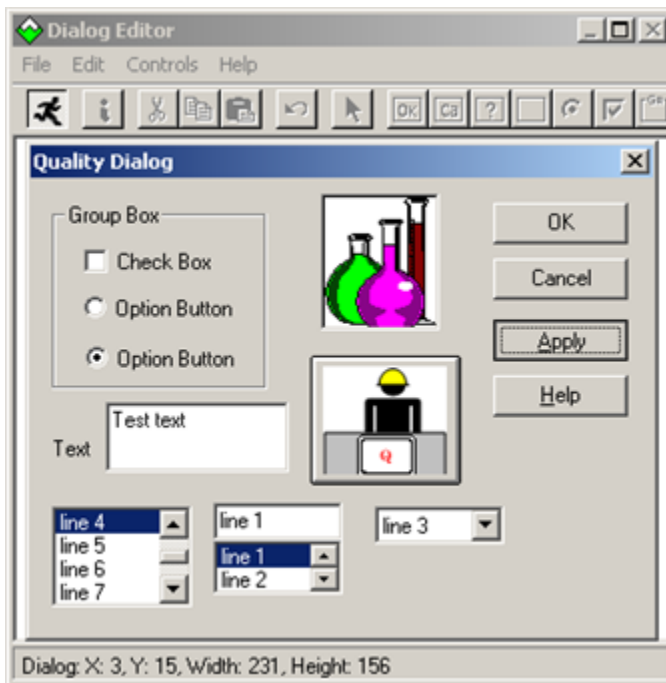
Result: The dialog box goes into Run mode.

Test the Dialog Box

When the dialog box is in Run mode the dialog box:

- Editing functionality is disabled.
- Controls perform the basic generic operations they are designed to do, e.g. the buttons go down and up; text can be written in a Text Box; scroll bars for multiple line controls scroll; check boxes and Option buttons can be checked.

Test the control on the running dialog box to make sure they are functioning correctly.



Check the following features.

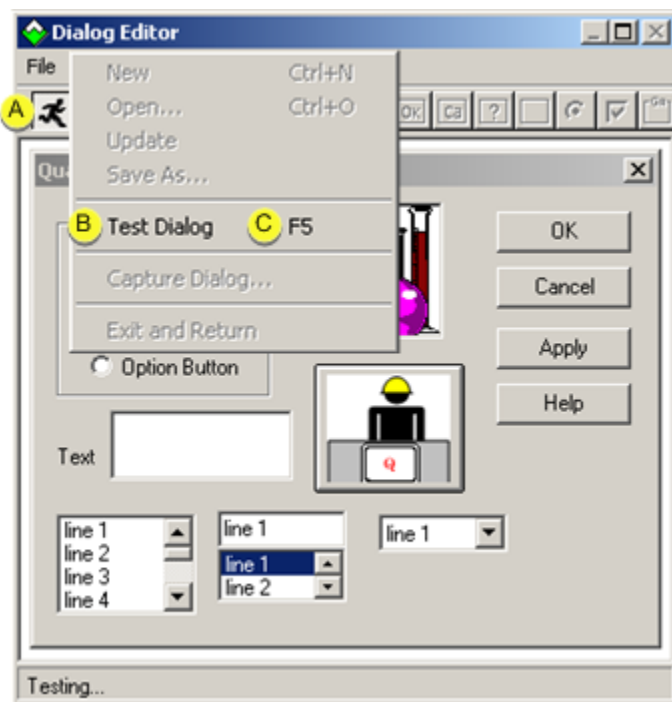
If a statement is False, stop the dialog box and make the required corrections.

Statement	T/F
-----------	-----

The Tab key moves through the control selection in a logical order. Note: Objects that users cannot act on, e.g. Group boxes, text controls and pictures are not selected.	
The Option buttons are grouped correctly.	
Text in a text box wraps or does not wrap, according to requirements.	
All of the accelerator keys work correctly.	

Stop the test

Do one of the following.



A	Click the Test Dialog button
B	Click File>Test Dialog on the Dialog Editor menu bar.
C	Press F5 on the keyboard.

Result: The dialog box goes back to Edit mode.

7. Exit from the Dialog Editor

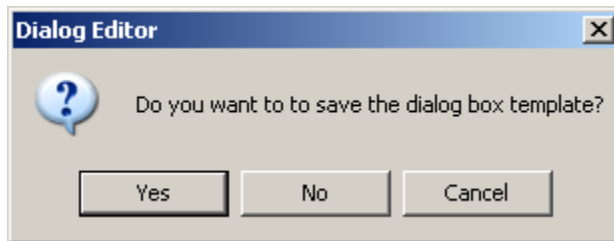
1. Do one of the following.



A	Click File>Exit and Return on the Dialog Editor menu bar.
B	Click the Exit button on the Dialog Editor title bar.
C	Press Alt+F4 on the keyboard.

A message opens with the following question.

Do you want to save the dialog box template?



2. Do one of the following.

- Click Yes.

Results:

- a. The Dialog Editor closes.
- b. The dialog box template code is inserted into the Program Editor script.

Important: Any highlighted line or lines in the script will be overwritten.

- Click No.

The Dialog Editor closes.

- Click Cancel.

The Exit command is cancelled; the Dialog Editor remains opens.

8. Use a Custom Dialog Box in a Script

8. Use a Custom Dialog Box in a Script

After using Dialog Editor to insert a custom dialog box template into your script, you'll need to make the following modifications to your script:

8.1 (<i>on page 208</i>)	Create a dialog record.
8.2 (<i>on page 209</i>)	Enter information into the custom dialog box.
8.3 (<i>on page 211</i>)	Display the custom dialog box.
8.4 (<i>on page 212</i>)	Retrieve values from the custom dialog box.

8.1. Create a Dialog Record

To store the values retrieved from a custom dialog box, you create a dialog record with a **Dim** statement, using the following syntax:

```
Dim DialogRecord As DialogVariable
```

Examples

Following are examples of how to create dialog records

Dim b As UserDialog	'Define a dialog record "b".
---------------------	------------------------------

Dim PlayCD As CDDialog	'Define a dialog record "Play-CD".
------------------------	------------------------------------

Sample script

This sample script that illustrates how to create a dialog record named **b** within a dialog box template named **UserDialog**. Notice that the order of the statements within the script is as follows: the dialog box template precedes the statement that creates the dialog record, and the **Dialog** statement follows both of them.

```
Sub Main()

  Dim ListBox1$()      'Initialize list box array.

  'Define the dialog box template.

  Begin Dialog UserDialog , ,163,94,"Grocery Order"

    Text 13,6,32,8,"&Quantity:",.Text1

    TextBox 48,4,28,12,.TextBox1

    ListBox 12,28,68,32,ListBox1$,.ListBox1

    OKButton 112,8,40,14

    CancelButton 112,28,40,14

  End Dialog

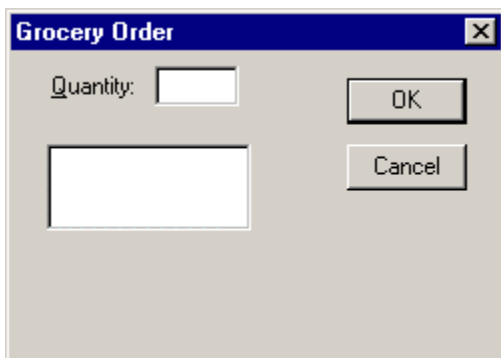
  Dim b As UserDialog  'Create the dialog record.

  Dialog b             'Display the dialog box.

End Sub
```

8.2. Enter Information into the Custom Dialog Box

If you open and run the sample script shown in the preceding subsection, you will see a dialog box that resembles the following.



- Controls to which values can be assigned.
- Define and fill an array.
- Set default text in a text box.
- Set the initial focus and controlling the tabbing order.

Controls to which Values can be Assigned

This custom dialog box is not very useful. For one thing, the user doesn't see any items in the list box along the left side of the dialog box. To put information into this dialog box, you assign values to its controls by modifying the statements in your script that are responsible for displaying those controls to the user. The following table lists the dialog box controls to which you can assign values and the types of information you can control:

Control(s)	Types of Information
List box, drop list box, and combo box	Items
Text box	Default text
Check box	Values

Define and Fill an Array

You can store items in the list box shown in the example above by creating an array and then assigning values to the elements of the array. For example, you could include the following lines to initialize an array with three elements and assign the names of three common fruits to these elements of your array:

```
Dim ListBox1$(3)      'Initialize list box array.

ListBox1$(0) = "Apples"

ListBox1$(1) = "Oranges"

ListBox1$(2) = "Pears"
```

Set Default Text in a Text Box

You can set the default value of the text box in your script to 12 with the following statement, which must follow the statement that defines the dialog record but precede the statement or function that displays the custom dialog box:

```
b.TextBox1 = "12"
```

Set the Initial Focus and Controlling the Tabbing Order

You can determine which control has the focus when your custom dialog box is first displayed as well as the tabbing order between controls by understanding two rules that the Basic Control Engine script follows. First, the focus in a custom dialog box is always set initially to the first control to appear in the dialog box template. Second, the order in which subsequent controls appear within the dialog box template determines the tabbing order. That is, pressing the **Tab** key will change the focus from the first control to the second one, pressing the **Tab** key again will change the focus to the third control, and so on.

8.3. Display the Custom Dialog Box

To display a custom dialog box, you can use either of the following.

Dialog() function
Dialog statement.

Dialog() Function

You can use a **Dialog()** function to determine how the user closed your custom dialog box. For example, the following statement will return a value when the user clicks an **OK** button or a **Cancel** button or takes another action:

```
response% = Dialog(b)
```

The **Dialog()** function returns any of the following values:

Value returned	If a user clicks:
-1	The OK button.
0	The Cancel button.
>0	A push button. The returned number represents which button was clicked based on its order in the dialog box template (1 is the first push button, 2 is the second push button, and so on).

Dialog Statement

You can use the **Dialog** statement when you don't need to determine how the user closed your dialog box. You'll still be able to retrieve other information from the dialog box record, such as the value of a list box or other dialog box control.

An example of the correct use of the **Dialog** statement is:

Dialog mydlg

Where

Dialog is the statement that calls a declared dialog name.

mydlg is the dialog name in this example.

8.4. Retrieve Values from the Custom Dialog Box

After displaying a custom dialog box for your user, your script must retrieve the values of the dialog controls. You retrieve these values by referencing the appropriate identifiers in the dialog record.

You'll find an example of how to retrieve values from a custom dialog box in the following subsection.

Sample

In the following sample script several of the techniques described earlier in this section have been used.

In this script, the array named **ListBox1** is filled with three elements (**"Apples"** , **"Oranges"** , and **"Pears"**). The default value of **TextBox1** is set to 12. A variable named **response** is used to store information about how the custom dialog box was closed. An identifier named **ListBox1** is used to determine whether the user chose **"Apples"** , **"Oranges"** , or **"Pears"** in the list box named **ListBox\$** . Finally, a **Select Case . . . End Select** statement is used to display a message box appropriate to the manner in which the user dismissed the dialog box.

```
Sub Main()

  Dim ListBox1$(2)      'Initialize list box array.

  Dim response%

  ListBox1$(0) = "Apples"

  ListBox1$(1) = "Oranges"

  ListBox1$(2) = "Pears"

  Begin Dialog UserDialog , ,163,94,"Grocery Order"

    Text 13,6,32,8,"&Quantity:",.Text1 'First control in
        'template gets the focus.

    TextBox 48,4,28,12,.TextBox1

    ListBox 12,28,68,32,ListBox1$,.ListBox1

    OKButton 112,8,40,14

    CancelButton 112,28,40,14
```

```

End Dialog

Dim b As UserDialog 'Create the dialog record.

b.TextBox1 = "12" 'Set default value of the text box
                'to 1 dozen.

response = Dialog(b) 'Display the dialog box.

Select Case response%

Case -1

    Fruit$ = ListBox1$(b.ListBox1)

    MsgBox "Thank you for ordering " + b.TextBox1 + " " + Fruit$ + "."

Case 0

    MsgBox "Your order has been canceled."

End Select

End Sub

```

9. Use a Dynamic Dialog Box in a Script

9. Use a Dynamic Dialog Box in a Script

You can retrieve values from a custom dialog box while the dialog box is displayed, using the dynamic dialog boxes feature.

9.1 (on page 213)	Sample script for a dynamic dialog box.
9.2 (on page 215)	Make a dialog box dynamic.

9.1. Sample Script for a Dynamic Dialog Box

The following script illustrates the most important concepts you'll need to understand in order to create a dynamic dialog box in your script:

```

'Dim "Fruits" and "Vegetables" arrays here to make them accessible to
'all procedures.

Dim Fruits(2) As String

Dim Vegetables(2) As String

'Dialog procedure--must precede the procedure that defines the custom

```

```

'dialog box.

Function DialogControl(ctrl$, action%, suppvale%) As Integer

Select Case action%

Case 1

    DlgListBoxArray "ListBox1", fruits 'Fill list box with

        'items before dialog

        'box is visible.

    DlgValue "ListBox1", 0 'Set default value to

        'first item in list box.

Case 2

    'Fill the list box with names of fruits or vegetables

    'when the user selects an option button.

    If ctrl$ = "OptionButton1" Then

        DlgListBoxArray "ListBox1", fruits

        DlgValue "ListBox1", 0

    ElseIf ctrl$ = "OptionButton2" Then

        DlgListBoxArray "ListBox1", vegetables

        DlgValue "ListBox1", 0

    End If

End Select

End Function

Sub Main()

Dim ListBox1$() 'Initialize array for use by ListBox

    'statement in template.

Dim Produce$

'Assign values to elements in the "Fruits" and "Vegetables"

'arrays.

Fruits(0) = "Apples"

Fruits(1) = "Oranges"

Fruits(2) = "Pears"

Vegetables(0) = "Carrots"

Vegetables(1) = "Peas"

Vegetables(2) = "Lettuce"

'Define the dialog box template.

Begin Dialog UserDialog , ,163,94,"Grocery Order",.DialogControl

Text 13,6,32,8,"&Quantity:",.Text1 'First control in

    'template gets the focus.

```

```

TextBox 48,4,28,12,.TextBox1

ListBox 12,28,68,32,ListBox1$,.ListBox1

OptionGroup .OptionGroup1

  OptionButton 12,68,48,8,"&Fruit",.OptionButton1

  OptionButton 12,80,48,8,"&Vegetables",.OptionButton2

OKButton 112,8,40,14

CancelButton 112,28,40,14

End Dialog

im b As UserDialog  'Create the dialog record.

b.TextBox1 = "12"  'Set the default value of the text
                  'box to 1 dozen.

response% = Dialog(b)  'Display the dialog box.

Select Case response%

Case -1

  If b.OptionGroup1 = 0 Then

    produce$ = fruits(b.ListBox1)

  Else

    produce$ = vegetables(b.ListBox1)

  End If

  MsgBox "Thank you for ordering " & b.TextBox1 & " " & produce$ & "."

Case 0

  MsgBox "Your order has been canceled."

End Select

End Sub

```

9.2. Make a Dialog Box Dynamic

The first thing to notice about the preceding script is that an identifier named `DialogControl` has been added to the `Begin Dialog` statement. As you will learn in the following subsection, this parameter to the `Begin Dialog` statement tells the Basic Control Engine to pass control to a function procedure named `DialogControl`.

- Use a dialog function.
- Respond to user actions.

Use a Dialog Function

Before the Basic Control Engine displays a custom dialog box by executing a **Dialog** statement or **Dialog()** function, it must first initialize the dialog box. During this initialization process, the Basic Control Engine checks to see whether you've defined a dialog function as part of your dialog box template. If so, the Basic Control Engine will give control to your dialog function, allowing your script to carry out certain actions, such as hiding or disabling dialog box controls.

After completing its initialization process, the Basic Control Engine displays your custom dialog box. When the user selects an item in a list box, clears a check box, or carries out certain other actions within the dialog box, the Basic Control Engine will again call your dialog function.

Responding to User Actions

The Basic Control Engine dialog function can respond to six types of user actions:

Ac-tion	Description
1	This action is sent immediately before the dialog box is shown for the first time.
2	This action is sent when:
	A button is clicked, such as OK , Cancel , or a push button.
	A check box's state has been modified
	An option button is selected. In this case, <code>ControlName\$</code> contains the name of the option button that was clicked, and <code>SuppValue</code> contains the index of the option button within the option button group (0 is the first option button, 1 is the second, and so on).
	The current selection is changed in a list box, drop list box, or combo box. In this case, <code>Control-Name\$</code> contains the name of the list box, combo box, or drop list box, and <code>SuppValue</code> contains the index of the new item (0 is the first item, 1 is the second, and so on).
3	This action is sent when the content of a text box or combo box has been changed and that control loses focus.
4	This action is sent when a control gains the focus.
5	This action is sent continuously when the dialog box is idle. The user should return a 0 or idle processing will use up the CPU.
6	This action is sent when the dialog box is moved.

You'll find a more complete explanation of these action codes in the A–Z Reference. See the **DlgProc** (Function) entry in that documentation.

Debug Scripts

Debug Scripts

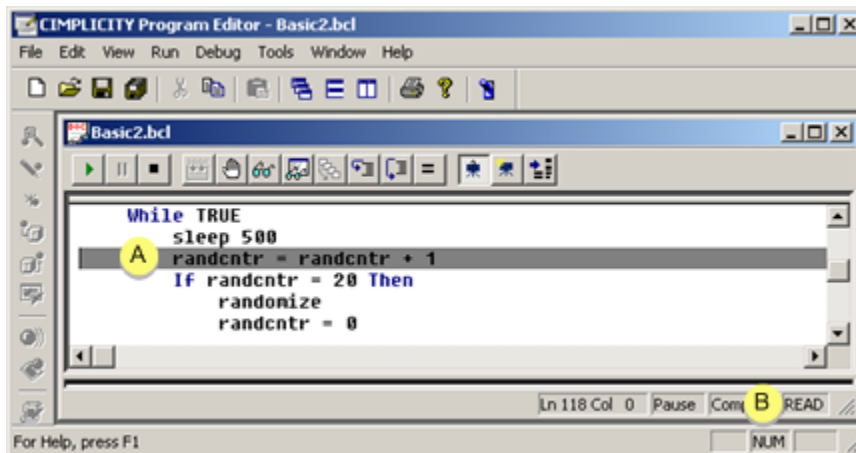
- Debug overview
- Debug options

Debug Overview

While debugging, you are actually executing the code in a script line by line.

1. Start the script.
2. Click the Pause button  on the Application toolbar.

The script is ready to be debugged.



A	The Program Editor displays an instruction pointer on the line of code that is about to be executed. When the instruction pointer is on a line of code, the text on that line appears in black on a gray background that spans the width of the entire line.
B	The edit pane is read-only during the debugging process. You are free to move the insertion point throughout the script, select text and copy it to the Clipboard as necessary, set break-points, and add and remove watch variables, but you cannot make any changes to the script until you stop running it.

Debug Options

1 (on page 218)	Fabricate event information.
2 (on page 220)	Step through scripts.
3 (on page 225)	Use breakpoints.
4 (on page 229)	Perform traces in scripts.
5 (on page 232)	Use a Watch variable.

1. Fabricate Event Information

The Event Editor allows the user to configure a script to run in response to an event. When a project is running, the Event Manager runs a script either when a specified event or any event occurs, depending on what is specified in the script.

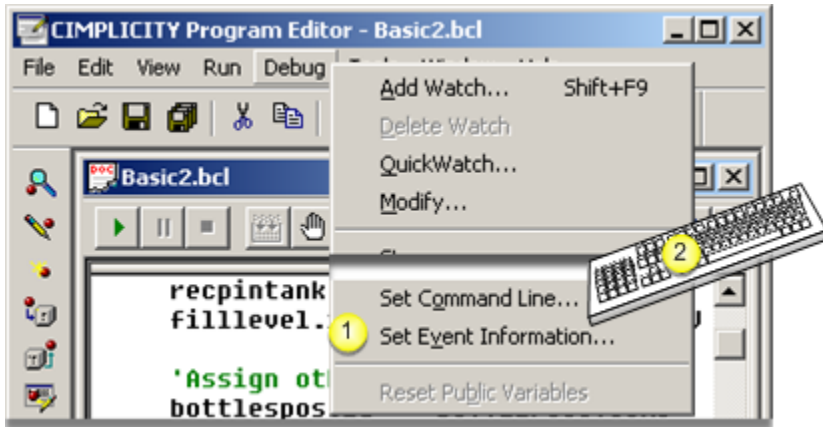
However

- On one hand, when you build the script in the Program Editor there is no real event to trigger the script.
- On the other hand, when the script runs with the Event Manager, you can not debug it in the Program Editor.

An Event Information dialog box is available to fabricate the event information that the APIs would provide in a real environment.

Using this fabrication you can safely test the accuracy of your script.

Do one of the following.



- | | |
|---|--|
| 1 | Select Debug>Set Event Information on the Program Editor menu bar. |
| 2 | Press D+Alt+V on the keyboard. |

Result: The Event Information dialog box opens.

 A screenshot of the "Event Information" dialog box. It has a blue title bar and a close button. The dialog is divided into two main sections. The left section contains:

- Event Type: Alarm Generated (dropdown)
- Event ID: EVENT1 (text field)
- Action ID: ACTION1 (text field)
- Object ID: TANKTEMP (text field with a search icon)
- Point section: Point ID (text field with a search icon), Available (checkbox), Value (text field)

 The right section, titled "Alarm", contains:

- Alarm ID: S90_900 (text field with a search icon)
- Resource ID: 90-70 (text field with a search icon)
- Alarm Class: HIGH (text field with a search icon)
- Message: CONTACT SUPERVISOR (text field)
- Ref Id: 10 (text field)
- Prev State: Acknowledged (dropdown)
- Final State: Acknowledged (dropdown)
- Req: Acknowledged (dropdown)

 At the bottom are "OK" and "Cancel" buttons.

Because you are using the Event Information dialog box to fabricate a real world environment, what you enter depends on what is included in your script. If the script includes specific entries for any of the fields, you have to enter what is in the script. For any entries that are not specifically referred to in the script, you can enter whatever you want.

Example

A script defines the:

Event Type as Alarm Generated

Resource ID as **\$SYSTEM** .

As a result, **Alarm Generated** and **\$SYSTEM** are selected in the Event Information dialog box.

Entries in the other fields are fictitious.

Whenever, you run the script, it will draw its information from what you entered. You only have to change your entries in the Event Information dialog box if a script requires a change in a specific entry.

2. Step through Scripts

- Step through a script procedure
- Step through a script tools

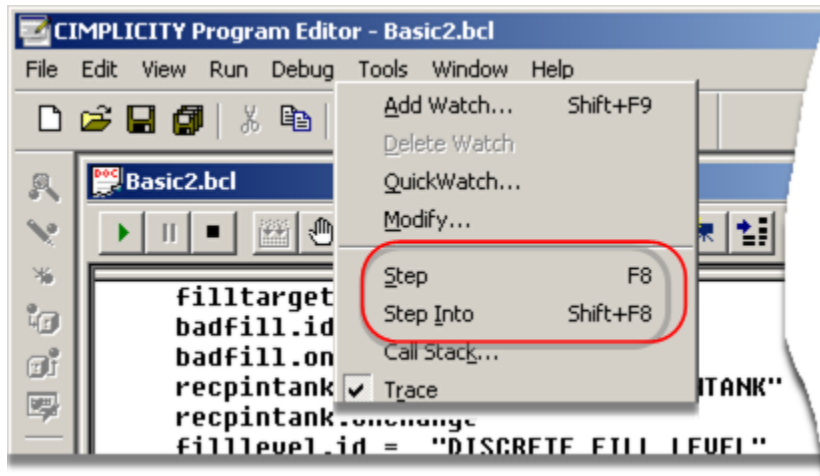
Step through a Script Procedure

Two methods are available to step through a script.

Both methods involve stepping through a script code line by line.

Method	Description
Single step	Steps into calls to user-defined functions and subroutines.
Procedure step	Does not step into calls to user defined functions and subroutines. The procedure step does execute the calls.

1. Use either method to start stepping through your script with either the single step or procedure step method.



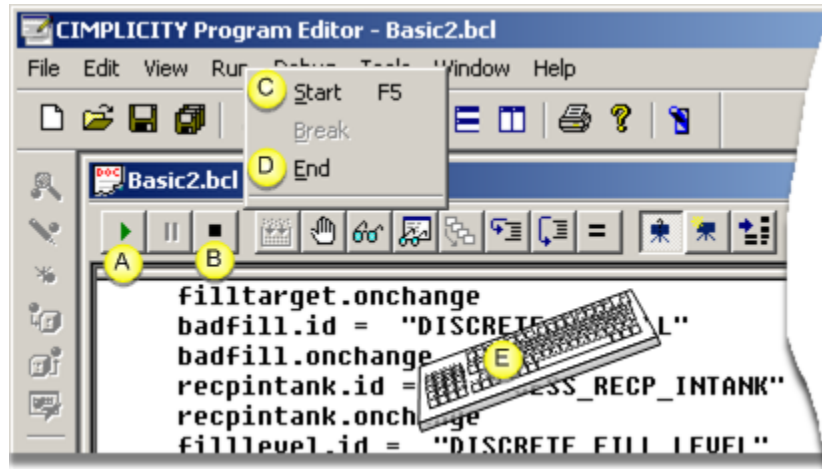
Method	Do one of the following.
Single step	<ul style="list-style-type: none"> ◦ Click Debug>Step on the CIMPLICITY Program Editor menu bar. ◦ Press F8 on the keyboard.
Procedure step	<ul style="list-style-type: none"> ◦ Click Debug>Step Into on the CIMPLICITY Program Editor menu bar. ◦ Press Shift + F8 on the keyboard.

The Program Editor places the instruction pointer on the `sub main` line of the script.

2. Repeat the command as many times as necessary to continue stepping through..

Each time you repeat the Step command, Program Editor executes the line containing the instruction pointer and moves the instruction pointer to the next line to be executed.

3. Do one of the following when you finish stepping through the script execution.



A	Click the Start button on the Application toolbar.
B	Click the End button on the Application toolbar.
C	Click Run>Start on the Program Editor menu bar.
D	Click Run>End on the Program Editor menu bar.
E	Press F5 on the keyboard.

**Note:**

When script execution is initiated, the script will first be compiled, if necessary. Therefore, there may be a slight pause before execution actually begins. If the script contains any compile errors, it will not be executed.

Do the following.

4. Correct any compile errors
5. Initiate execution again.
6. Repeat the Step command to continue stepping through the script line by line,

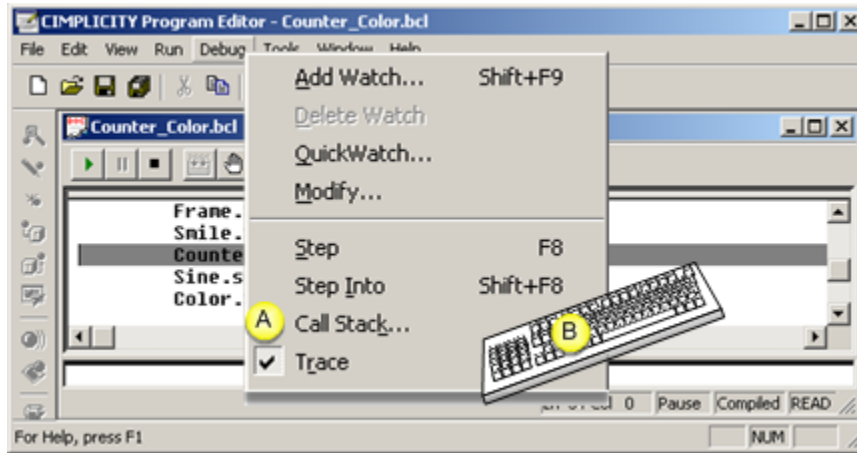
Step through a Script Tools

Calls dialog box
Set Next Statement

Calls dialog box

When stepping through a subroutine, you can determine the procedure calls made to arrive at the paused point in the script..

7. Do one of the following.

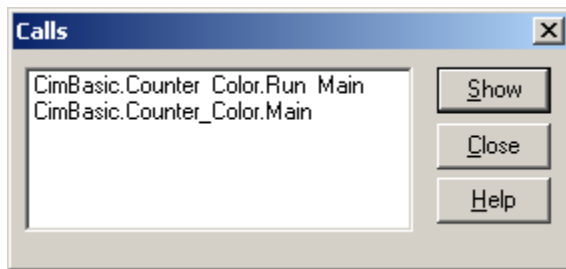


A	Click Debug>Call Stack on the Program Editor menu bar.
B	Press Alt+D+K on the keyboard.

The Calls dialog box opens when you use either method.

The Calls dialog box lists the procedure calls made by the script to arrive at the selected subroutine.

8. Do one of the following.



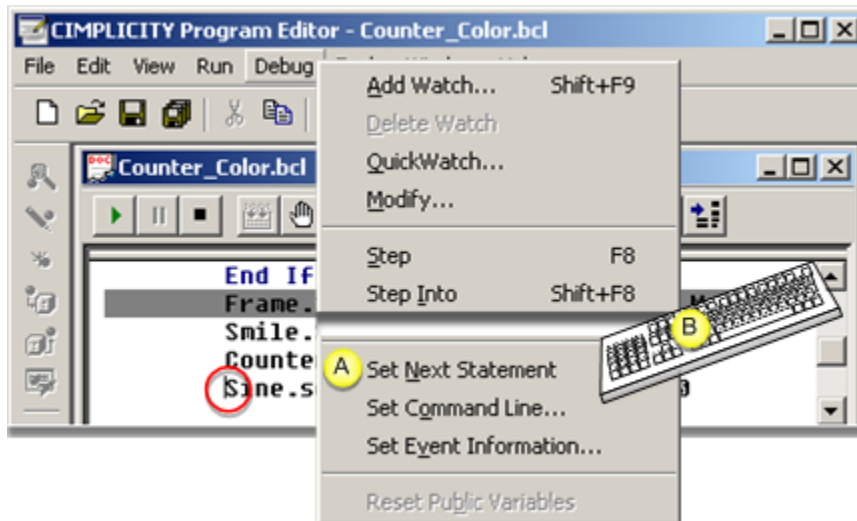
Click	Description
Show	a. Select a procedure call in the Calls list. b. Click Show.

	<ul style="list-style-type: none"> ◦ The Calls dialog box closes. ◦ The Program Editor highlights the currently executing line in the procedure you selected, scrolling that line into view if necessary.
Close	Closes the Calls dialog box.
Help	Opens the Program Editor documentation.

Set Next Statement

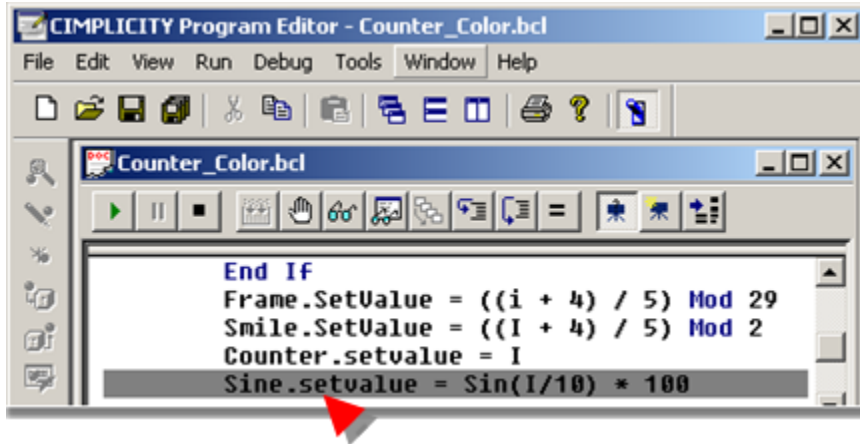
When stepping through a subroutine, you can move the insertion point to another line within a subroutine to repeat or skip execution of a section of code.

- Place the insertion point in the line where you want to resume stepping through the script.
- Do one of the following.



A	Click Debug>Set Next Statement on the Program Editor menu bar.
B	Press Alt+D+N on the keyboard.

The instruction pointer moves to the line you selected; you can resume stepping through your script from there.

**Note:**

You can only use the Set Next Statement command to move the instruction pointer within the same subroutine.

3. Use Breakpoints

1 (on page 226)	Select breakpoints.
2 (on page 226)	Run the Debugger.
3 (on page 228)	Remove breakpoints.

If you only need to debug one or more portions of a long script one or more breakpoints can be set at selected lines in your script.

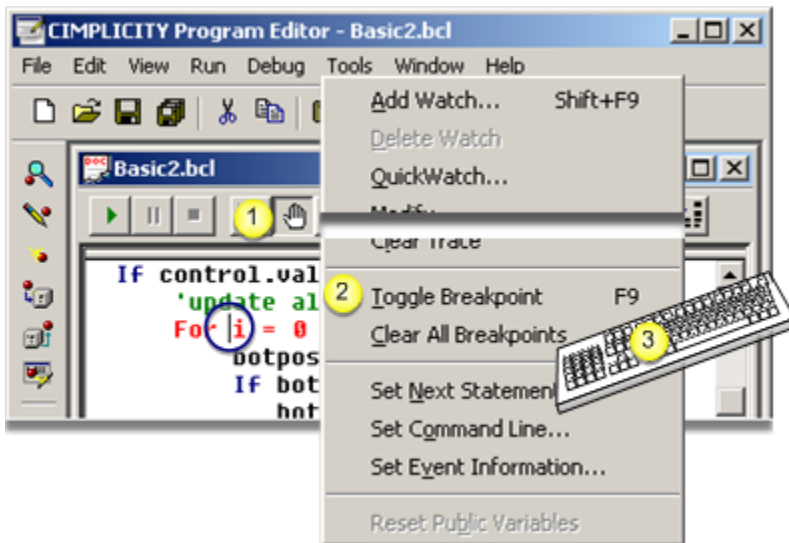
Valid breakpoints can only be set on lines in your script that contain code, including lines in functions and subroutines. Although you can set a breakpoint anywhere within a script prior to execution, when you compile and run the script, invalid breakpoints (breakpoints on lines that don't contain code) are automatically removed. While you are debugging your script, the Program Editor will beep if you try to set a breakpoint on a line that does not contain code.

Select Breakpoints

1. Place the insertion point in the line where you want to do one of the following.

- A select point.
- A line outside the current subroutine.
- Selected portions of a program.

1. Do one of the following.



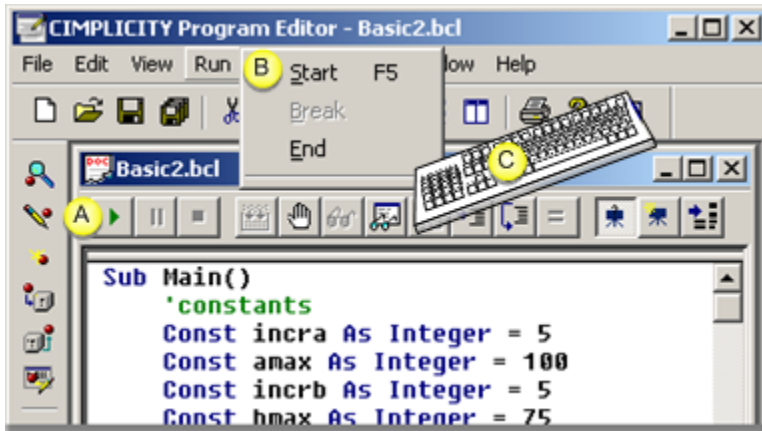
1	Click the Breakpoint button on the Application toolbar.
2	Click Debug>Toggle Breakpoint on the Program Editor menu bar.
3	Press F9 on the keyboard.

1. Repeat the process to set as many breakpoints as needed, up to 255.

Result: The script is ready to be debugged.

Run the Debugger

Do any of the following.



A	Click the Start button on the Application toolbar.
B	Click Run>Start on the Program Editor menu bar.
D	Press F5 on the keyboard.

Results

A breakpoint was inserted at:

- A select point

The Program Editor:

Runs the script at full speed until it reaches the line containing the breakpoint and then pauses with the instruction pointer on that line.

The line will be executed next when you either proceed with debugging or resume running the script.

If you want to continue debugging at another line in your script, you can use the **Set Next Statement** command in the Debug menu to move the instruction pointer to the desired line—provided the line is within the same subroutine.

- A line outside the current subroutine

Runs the script at full speed until it reaches the line containing the breakpoint and then pauses with the instruction pointer on that line.

You can now resume stepping through your script from that point.

- Selected portions of a program

Runs the script at full speed until it reaches the line containing the first breakpoint and then pauses with the instruction pointer on that line.

1. Step through as much of the code as you need to.
2. To resume running your script, click the Start button on the toolbar or press F5.

The script executes at full speed until it reaches the line containing the second breakpoint and then pauses with the instruction pointer on that line.

1. Repeat 1 and 2 until you have finished debugging the selected portions of your script.

Remove breakpoints

Breakpoints can be removed either manually or automatically.

- Remove a single breakpoint manually

1. Place the insertion point on the line containing the breakpoint that you want to remove.
2. Do one of the following.

- Click the **Toggle Breakpoint** button on the Application toolbar.
- Press F9 on the keyboard.

Result: The breakpoint is removed, and the line no longer displays in contrasting type.

- Remove all breakpoints manually

Click Debug>Clear All Breakpoints on the Program Editor menu bar.

Result: The Program Editor removes all breakpoints from your script.



Note:

Breakpoints are removed automatically under the following circumstances:

- When a script is compiled and executed, breakpoints are removed from lines that don't contain code.
- When you exit from the Program Editor, all breakpoints are cleared.

4. Perform Traces in Scripts

The Trace command can be used in Basic Control Engine scripts to print output to the Program Editor window's Trace section.

1 (on page 229)	Enable tracing.
2 (on page 230)	Clear trace results.
3 (on page 231)	Disable tracing.

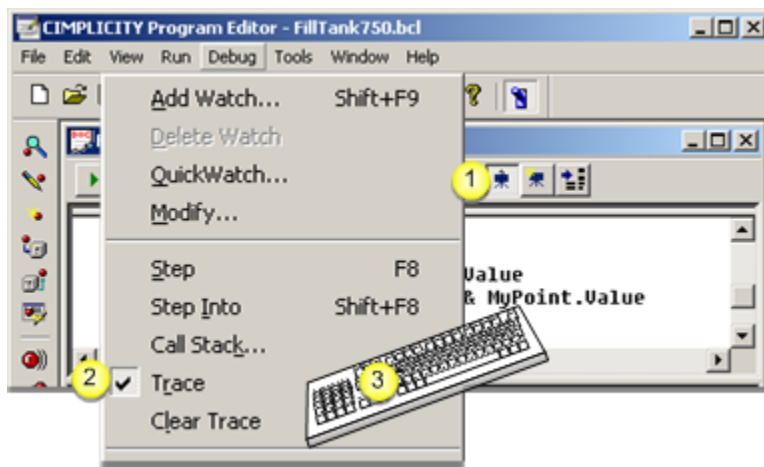
Enable Tracing

1. Enter a [Trace \(on page 922\)](#) command in a script.

Example

```
Trace "TANK750 " & MyPoint.Value
```

1. Do one of the following.



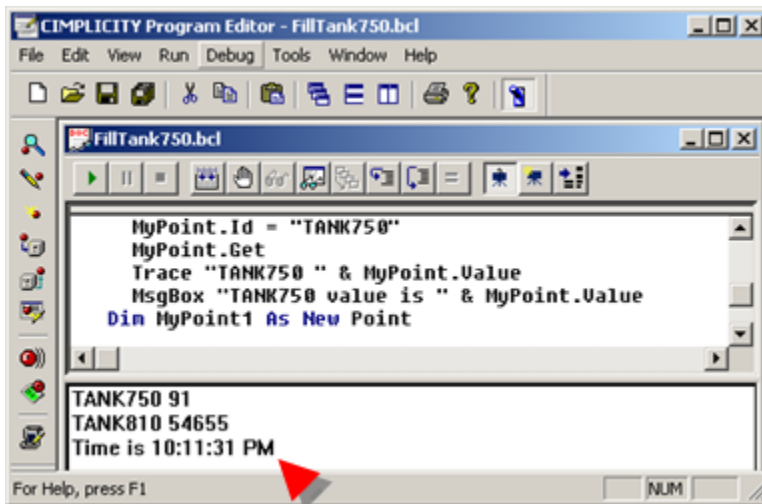
- | | |
|---|--|
| 1 | Depress the Trace button on the Application toolbar. |
|---|--|

- | | |
|---|---|
| 2 | Click Debug>Trace on the Program Editor menu bar. |
| 3 | Press Alt+D+R on the keyboard. |

Trace is enabled.

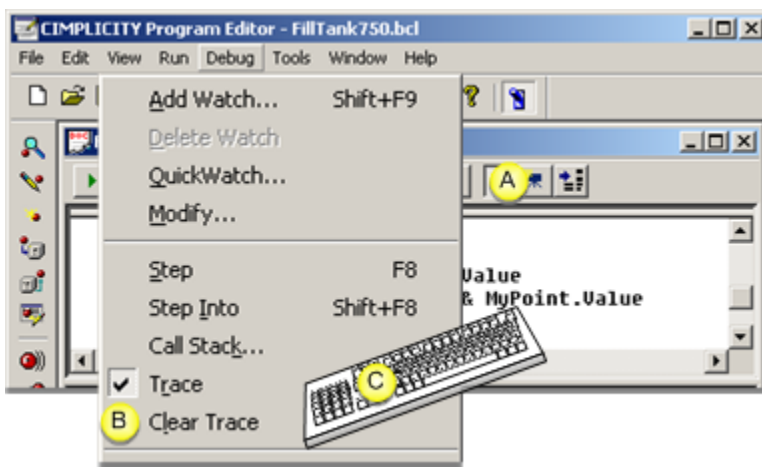
1. Run the script.

Result: The trace results display in the Program Editor window trace section.



Clear Trace Results

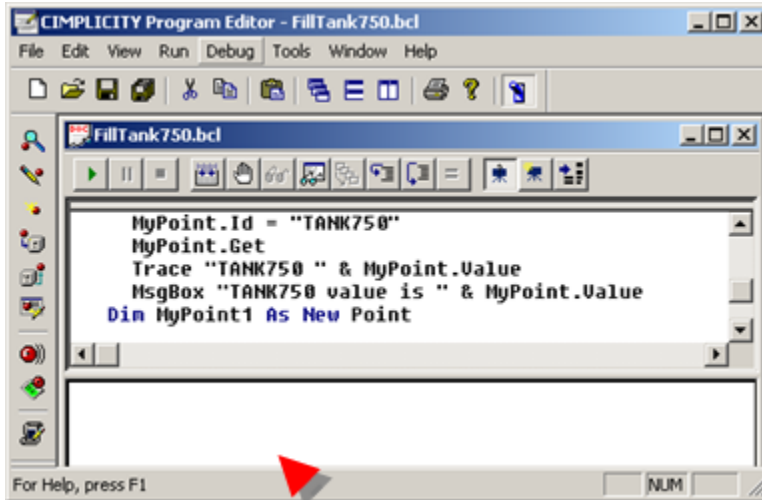
Do one of the following.



- | | |
|---|--|
| A | Click the Clear Trace button on the Application toolbar. |
|---|--|

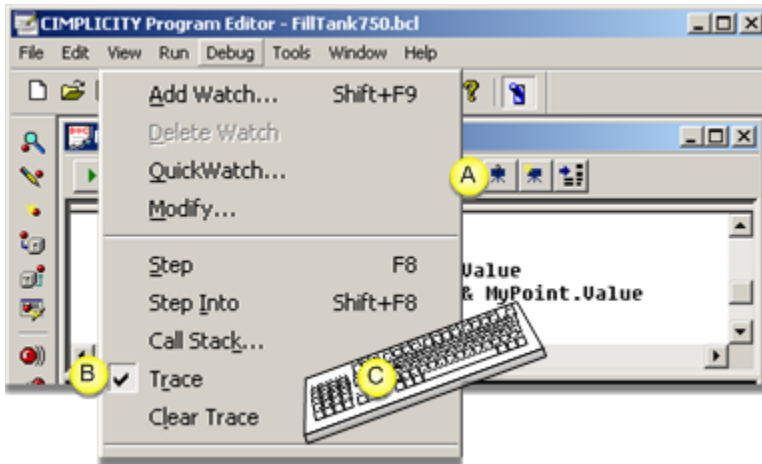
B	Click Debug>Clear Trace on the Program Editor menu bar.
C	Press Alt+D+L on the keyboard.

Result: The trace results are deleted from the Program Editor trace section.



Disable Tracing

Do one of the following.



A	Restore the Trace button on the Application toolbar.
B	Click Debug>Trace on the Program Editor menu bar.

C	Press Alt+D+R on the keyboard.
---	--------------------------------

Result: The trace entries are ignored when the script is run.

5. Use a Watch Variable

5. Use a Watch Variable

As you debug your script, you can use Program Editor's watch pane to monitor selected variables. For each of the variables on this watch variable list, Program Editor displays the name of the variable, where it is defined, its value (if the variable is not in scope, its value is shown as <not in context>), and other key information such as its type and length (if it is a string). The values of the variables on the watch list are updated each time you enter break mode.

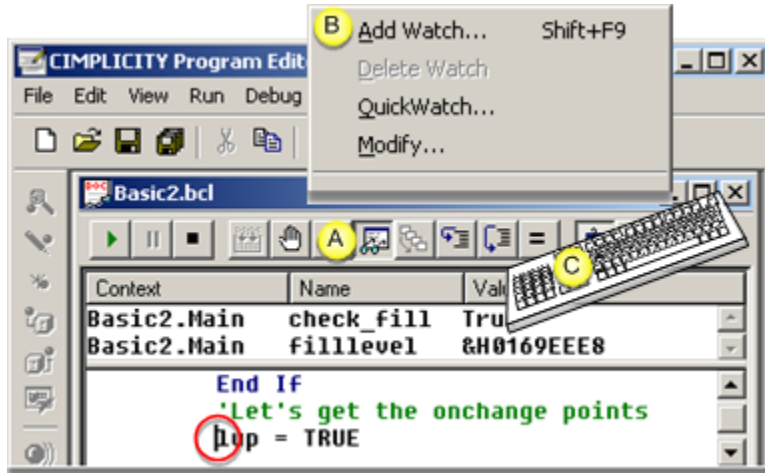
5.1 (on page 232)	Add a Watch variable to the Program Editor's Watch variable list.
5.2 (on page 235)	Modify the value of a Watch variable.
5.3 (on page 238)	Use Quick Watch.
5.4 (on page 240)	Delete a Watch variable.

5.1. Add a Watch Variable to the Program Editor's Watch Variable List

- [Select Variable Procedure](#)
- [Guidelines for Variables](#)

Select Variable Procedure

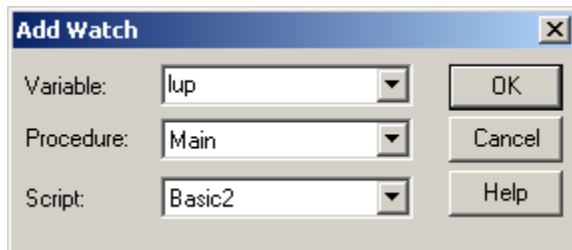
1. Select a variable in a script.
2. Do one of the following.



A	Click the Add Watch button on the Application toolbar.
B	Click Debug>Add Watch on the Program Editor menu bar.
C	Press Shift+F9 on the keyboard.

An Add Watch dialog box opens.

3. Enter specifications as follows.

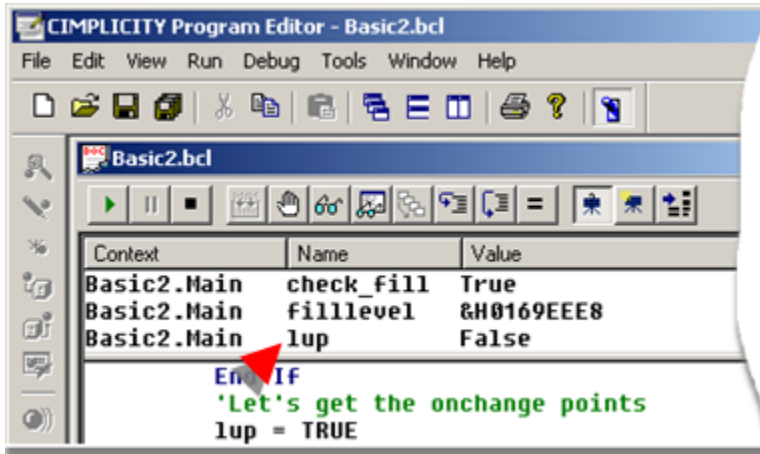


Field	Description
Variable	Name of the variable you want to add to the watch variable list.
Procedure	Procedure that will be watched.
Script	Script that will be watched.

4. Click OK or press Enter.

The selected Variable is added to the Add Watch list.

If this is the first variable you are placing on the watch variable list, the watch pane opens far enough to display that variable. If the watch pane was already open, it expands far enough to display the variable you just added.



guide:

Guidelines for Variables

- The following variables can or cannot be watched.

Cannot watch

Complex variables such as structures or arrays.

Can watch

- Variables of fundamental data types.

Examples

- Integer
- Long
- Variant
- Individual elements of arrays or structure members using the following syntax:

```
[variable [(index,...)] [.member [(index,...)]]...]
```

Where

<code>variable</code>	= Name of the structure or array variable,
<code>index</code>	= Literal number
<code>member</code>	= Name of a structure member.

Example

The following are valid watch expressions:

Watch Variable	Description
<code>a(1)</code>	Element 1 of array <code>a</code>
<code>person.age</code>	Member <code>age</code> of structure <code>person</code> .
<code>company(10,23).person.age</code>	Member <code>age</code> of structure <code>person</code> that is at element 10,23 within the array of structures named <code>company</code>

- If you are executing the script, you can
 1. Display the names of all the variables that are in scope or defined within the current function or subroutine on the drop-down Variable Name list.
 2. Select the variable you want from that list.
- You can add as many watch variables to the list as you want.

The Watch pane only expands until it fills half of Program Editor's application window. If your list of watch variables becomes longer than that, you can use the watch pane's scroll bars to bring hidden portions of the list into view.

- The list of watch variables is maintained between script executions.

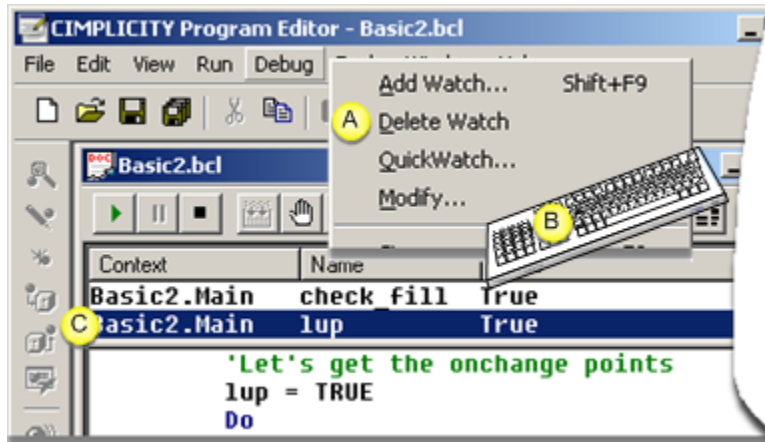
5.2. Modify the Value of a Watch Variable

When the debugger has [control \(on page 217\)](#), you can modify the value of any of the variables on Program Editor's Watch variable list.

- Procedure to modify variables.
- Guidelines for modifying variables.

Procedure to Modify Variables

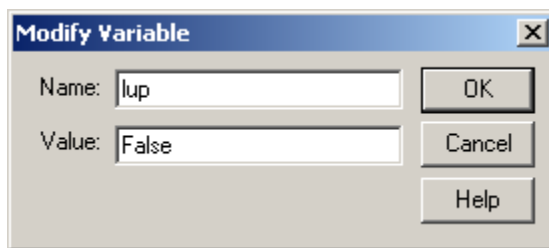
1. Select a variable to be modified.
2. Do one of the following.




A	Click Debug>Modify on the Program Editor menu bar.
B	Press Alt+D+M on the keyboard.
C	Double-click the variable line in the Watch list.

A Modify Variable dialog box opens.

3. Fill in the fields as follows.

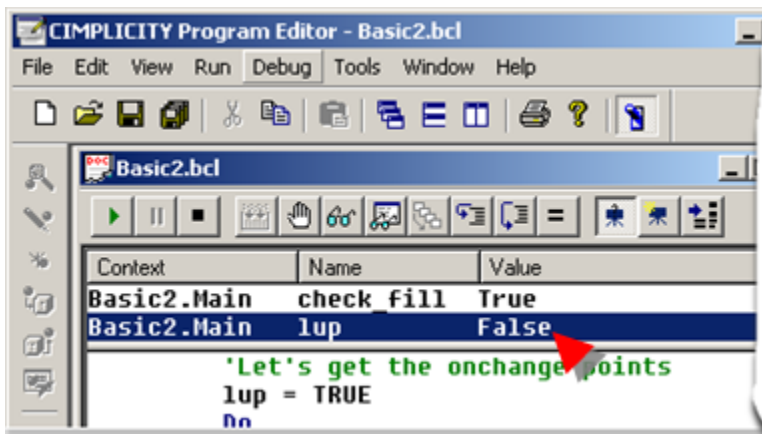


Field	Description
Name	Name of the variable to be modified. Note: If the line was double-clicked the Name field: <ul style="list-style-type: none"> ◦ Displays the selected variable. ◦ Is read-only.

	
Value	New value for the variable.

4. Click OK or press Enter.

The new variable value displays in the Watch list.



guide:

Guidelines for Modifying Variables

- When changing the value of a variable, the Program Editor converts the new value to be of the same type as the variable being changed.

Example

An Integer value is 3.

1.7 is entered in the Value field

The Program Editor converts the new value to 2.

- When modifying a `variant` variable, the Program Editor needs to determine both the type and value of the data. Program Editor uses the following logic in performing this assignment (in this order):

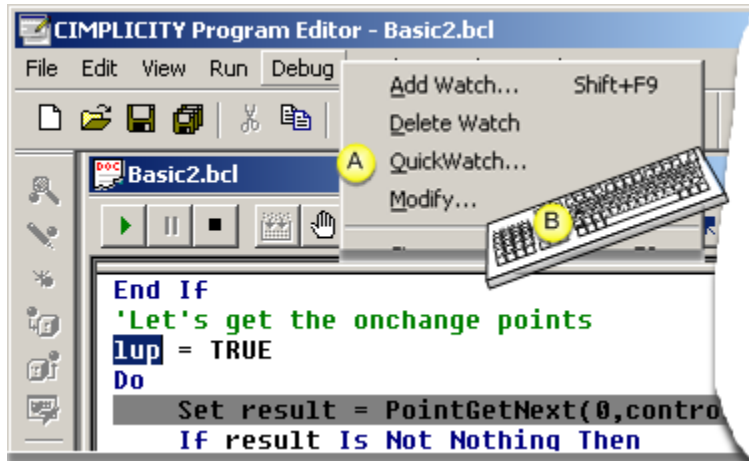
If the new value is	The variant variable is assigned:
Null	Null (VarType 1)
Empty	Empty (VarType 0).
True	True (VarType 11).
False	False (VarType 11).
number	The value of number. The type of the variant is the smallest data type that fully represents that number. You can force the data type of the variable using a type-declarator letter following number, such as %, #, &, !, or @.
date	The value of the new date (VarType 7)
Anything else	String (VarType 8).

- The Program Editor will not assign a new value if it cannot be converted to the same type as the specified variable.

5.3. Use Quick Watch

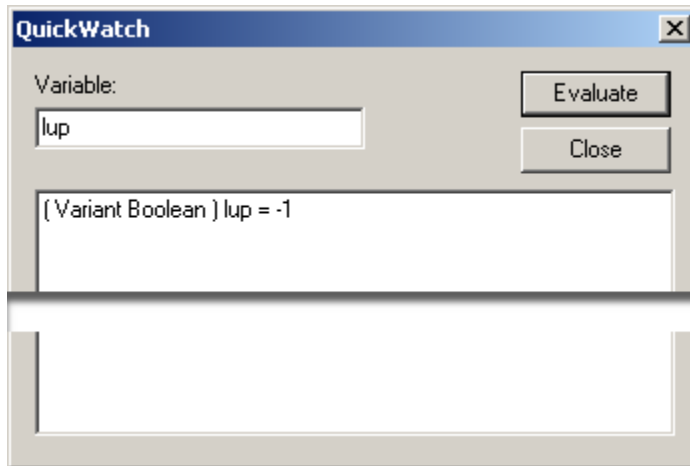
When the debugger has [control \(on page 217\)](#) , you can use the Quick Watch window to do a quick check of a variable value, without adding the variable to the [Watch list \(on page 232\)](#) .

1. Select the variable whose value you want to quickly check.
2. Do one of the following.



A	Click Debug>QuickWatch on the Program Editor menu bar.
B	Press Alt+D+Q on the keyboard.

The QuickWatch window opens displaying the value for the selected variable.



3. (Optional) Evaluate another variable.
 - a. Enter the variable in the Variable field.
 - b. Click Evaluate.

The variable is evaluated; if it has a known value, the value displays in the evaluation box.

4. Click Close.

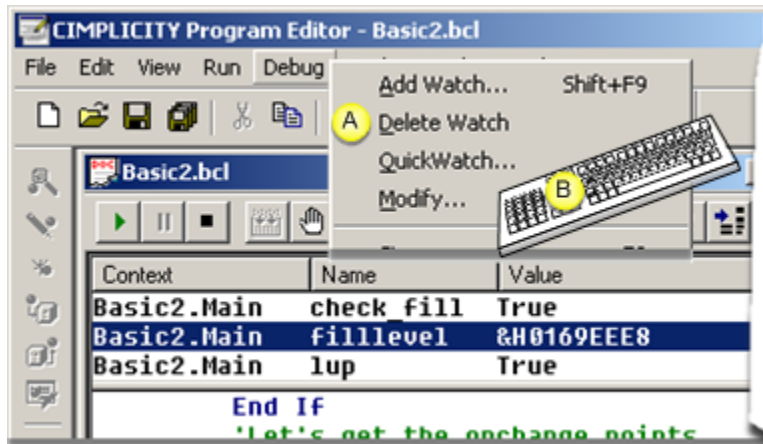
The QuickWatch window closes; you can continue debugging the script.

**Note:**

You must close the QuickWatch window in order to return to the script window.

5.4. Delete a Watch Variable

1. Select a variable on the Watch list.
2. Do one of the following.



A	Click Debug>Delete Watch on the Program Editor menu bar.
B	Press Alt+D+D on the keyboard.

The variable is deleted from the Watch list..

Run a Program

**Important:**

The CIMPLICITY project must be running in order to run the script.

Once you have finished editing your programs, you will want to run it to make sure it performs the way you intended. You can also suspend or stop an executing script.

Run a script

Note: This will also compile your script, if necessary, and then execute it.

- Click the Start button on the toolbar.
- Press F5.

The script is compiled (if it has not already been compiled), the focus is switched to the parent window, and the script is executed.

- Suspend A Running Program.
- Stop a running program.

Suspend a Running Program

Press Ctrl+Break. or click the Break toolbar button.

Execution of the script is suspended, and the instruction pointer (a gray highlight) appears on the line of code where the script stopped executing.



Note:

The instruction pointer designates the line of code that will be executed next if you resume running your script.

Stop a Running Program

Click the **End** tool on the toolbar.

Error Messages

Error Messages

This section contains listings of all the runtime errors. It is divided into two subsections, the first describing errors messages compatible with "standard" Basic as implemented by Microsoft Visual Basic and the second describing error messages specific to the Basic Control Engine.

A few error messages contain placeholders that get replaced by the runtime when forming the completed runtime error message. These placeholders appear in the following list as the italicized word placeholder.

1 (on page 242)	Visual Basic compatible error messages.
-----------------	---

2 (on page 245)	Basic Control Engine-specific error messages.
3 (on page 246)	Error message list.

1. Visual Basic Compatible Error Messages

The Visual Basic compatible error messages are:

Number	Message
3	Return without GoSub
5	Illegal procedure call
6	Overflow
7	Out of memory
9	Subscript out of range
10	This array is fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
19	No Resume
20	Resume without error
26	Dialog needs End Dialog or push button
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
49	Bad DLL calling convention
51	Internal error
52	Bad file name or number

53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
59	Bad record length
61	Disk full
62	Input past end of file
63	Bad record number
64	Bad file name
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable or With block variable not set
93	Invalid pattern string
94	Invalid use of Null
139	Only one user dialog may be up at any time
140	Dialog control identifier does not match any current control
141	The placeholder statement is not available on this dialog control type
143	The dialog control with the focus may not be hidden or disabled
144	Focus may not be set to a hidden or disabled control
150	Dialog control identifier is already defined
163	This statement can only be used when a user dialog is active

260	No timer available
281	No more DDE channels
282	No foreign application responded to a DDE initiate
283	Multiple applications responded to a DDE initiate
285	Foreign application won't perform DDE method or operation
286	Timeout while waiting for DDE response
287	User pressed Escape key during DDE operation
288	Destination is busy
289	Data not provided in DDE operation
290	Data in wrong format
291	Foreign application quit
292	DDE conversation closed or changed
295	Message queue filled; DDE message lost
298	DDE requires ddeml.dll
429	OLE Automation server can't create object
430	Class doesn't support OLE Automation
431	OLE Automation server cannot load file
432	File name or class name not found during OLE Automation operation
433	OLE Automation object does not exist
434	Access to OLE Automation object denied
435	OLE initialization error
436	OLE Automation method returned unsupported type
437	OLE Automation method did not return a value
438	Object doesn't support this property or method placeholder
439	OLE Automation argument type mismatch placeholder
440	OLE Automation error placeholder
443	OLE Automation Object does not have a default value

452	Invalid ordinal
460	Invalid Clipboard format
520	Can't empty clipboard
521	Can't open clipboard
600	Set value not allowed on collections
601	Get value not allowed on collections
603	ODBC - SQLAllocEnv failure
604	ODBC - SQLAllocConnect failure
608	ODBC - SQLFreeConnect error
610	ODBC - SQLAllocStmt failure
3129	Invalid SQL statement; expected 'DELETE', 'INSERT', 'PROCEDURE', 'SELECT', or 'UPDATE'
3146	ODBC - call failed
3148	ODBC - connection failed
3276	Invalid database ID

2. Basic Control Engine-Specific Error Messages

The Basic Control Engine-specific error messages are:

Number	Message
800	Incorrect Windows version
801	Too many dimensions
802	Can't find window
803	Can't find menu item
804	Another queue is being flushed
805	Can't find control
806	Bad channel number
807	Requested data not available

808	Can't create pop-up menu
809	Message box canceled
810	Command failed
811	Network error
812	Network function not supported
813	Bad password
814	Network access denied
815	Network function busy
816	Queue overflow
817	Too many dialog controls
818	Can't find list box/combo box item
819	Control is disabled
820	Window is disabled
821	Can't write to ini file
822	Can't read from ini file
823	Can't copy file onto itself
824	OLE Automation unknown object name
825	Can't re-dimension a fixed array
826	Can't load and initialize extension
827	Can't find extension
828	Unsupported function or statement
829	Can't find ODBC libraries
830	OLE Automation Lbound or Ubound on non-Array value
831	Incorrect definition for dialog procedure

3. Error Message List

The following table contains a list of all the errors generated by the Basic Control Engine compiler. With some errors, the compiler changes placeholders within the error to text from the script being compiled. These placeholders are represented in this table by the word placeholder.

Number	Message
1	Variable Required - Can't assign to this expression
2	Letter range must be in ascending order
3	Redefinition of default type
4	Out of memory, too many variables defined
5	Type-character doesn't match defined type
6	Expression too complex
7	Cannot assign whole array
8	Assignment variable and expression are different types
10	Array type mismatch in parameter
11	Array type expected for parameter
12	Array type unexpected for parameter
13	Integer expression expected for an array index
14	Integer expression expected
15	String expression expected
18	Left of "." must be an object, structure, or dialog
19	Invalid string operator
20	Can't apply operator to array type
21	Operator type mismatch
22	"placeholder" is not a variable
23	"placeholder" is not a array variable or a function
24	Unknown placeholder "placeholder"
25	Out of memory
26	placeholder: Too many parameters encountered

27	placeholder: Missing parameter(s)
28	placeholder: Type mismatch in parameter placeholder
29	Missing label "placeholder"
30	Too many nested statements
31	Encountered new-line in string
32	Overflow in decimal value
33	Overflow in hex value
34	Overflow in octal value
35	Expression is not constant
37	No type-characters allowed on parameters with explicit type
39	Can't pass an array by value
40	"placeholder" is already declared as a parameter
41	Variable name used as label name
42	Duplicate label
43	Not inside a function
44	Not inside a sub
46	Can't assign to function
47	Identifier is already a variable
48	Unknown type
49	Variable is not an array type
50	Can't redimension an array to a different type
51	Identifier is not a string array variable
52	0 expected
55	Integer expression expected for file number
56	placeholder is not a method of the object
57	placeholder is not a property of the object
58	Expecting 0 or 1

59	Boolean expression expected
60	Numeric expression expected
61	Numeric type FOR variable expected
62	For...Next variable mismatch
63	Out of string storage space
64	Out of identifier storage space
65	Internal error 1
66	Maximum line length exceeded
67	Internal error 3
68	Division by zero
69	Overflow in expression
70	Floating-point expression expected
72	Invalid floating-point operator
74	Single character expected
75	Subroutine identifier can't have a type-declaration character
76	Script is too large to be compiled
77	Variable type expected
78	Can't evaluate expression
79	Can't assign to user or dialog type variable
80	Maximum string length exceeded
81	Identifier name already in use as another type
84	Operator cannot be used on an object
85	placeholder is not a property or method of the object
86	Type-character not allowed on label
87	Type-character mismatch on routine placeholder
88	Destination name is already a constant
89	Can't assign to constant

90	Error in format of compiler extensions
91	Identifier too long
92	Expecting string or structure expression
93	Can't assign to expression
94	Dialog and Object types are not supported in this context
95	Array expression not supported as parameter
96	Dialogs, objects, and structures expressions are not supported as a parameter
97	Invalid numeric operator
98	Invalid structure element name following "."
99	Access value can't be used with specified mode
101	Invalid operator for object
102	Can't LSet a type with a variable-length string
103	Syntax error
104	placeholder is not a method of the object
105	No members defined
106	Duplicate type member
107	Set is for object assignments
108	Type-character mismatch on variable
109	Bad octal number
110	Bad number
111	End-of-script encountered in comment
112	Misplaced line continuation
113	Invalid escape sequence
114	Missing End Inline
115	Statement expected
116	ByRef argument mismatch

117	Integer overflow
118	Long overflow
119	Single overflow
120	Double overflow
121	Currency overflow
122	Optional argument must be Variant
123	Parameter must be optional
124	Parameter is not optional
125	Expected: Lib
126	Illegal external function return type
127	Illegal function return type
128	Variable not defined
129	No default property for the object
130	The object does not have an assignable default property
131	Parameters cannot be fixed length strings
132	Invalid length for a fixed length string
133	Return type is different from a prior declaration
134	Private variable too large. Storage space exceeded
135	Public variables too large. Storage space exceeded

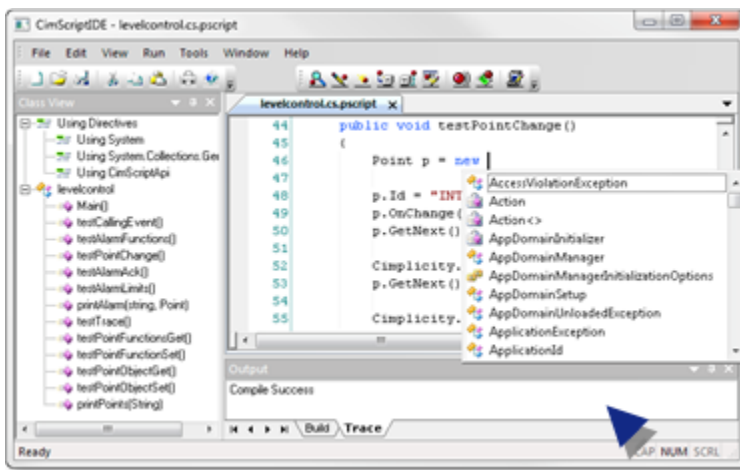
Chapter 3. CimScriptIDE Editor

About the CimScriptIDE Editor

A CimScriptIDE Editor enables and facilitates writing C# and VB .NET scripts.

An overview of how to open and take advantage of the CimScriptIDE editor includes the following.

1 (on page 266)	Open the CimScriptIDE editor.
2 (on page 256)	CimScriptIDE editor: Overview.
3 (on page 265)	Technical Reference: CimScriptIDE editor.



Important:

If you are familiar with the Program Editor for CimBasic, it is important to note that the CimScriptIDE Editor for .NET scripting behaves differently than the CimBasic Program Editor in regard to Compiling. Selecting [Compile \(on page 260\)](#) for a .NET script saves the script file to disk; selecting [Compile \(on page 116\)](#) for CimBasic it does not.

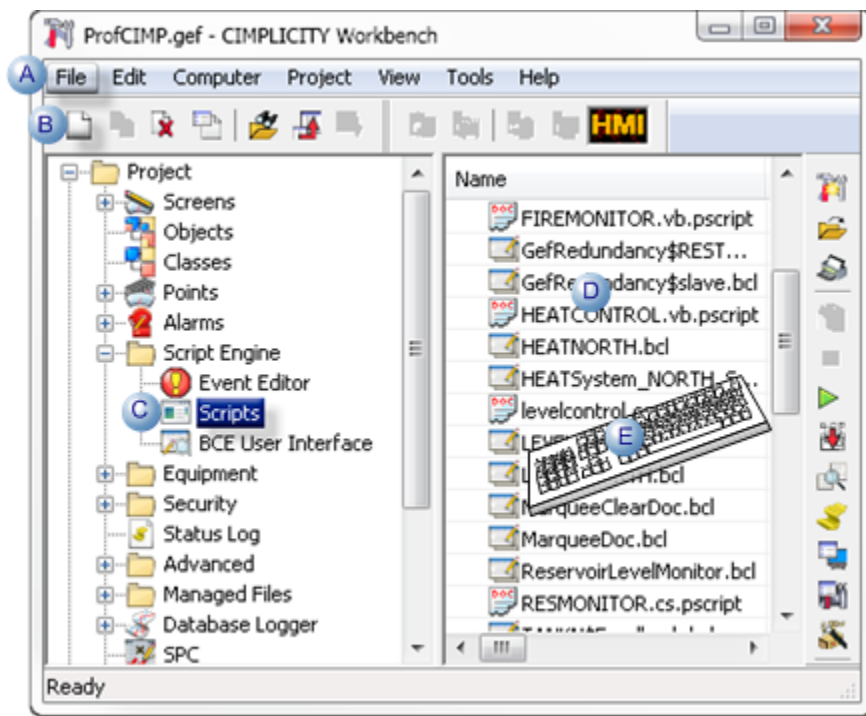
1. Open the CimScriptIDE Editor

1. Open the CimScriptIDE Editor

1.1 (on page 253)	Create a New C# or VB .NET script.
1.2 (on page 255)	Open an Existing C# or VB .NET Script

1.1. Create a New C# or VB .NET Script

1. Select **Project>Script Engine>Scripts** in the Workbench left pane.
2. Do one of the following.

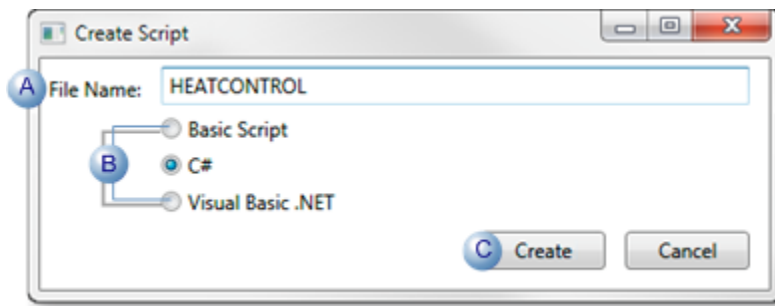


A	Click File>New>Object on the Workbench menu bar.
B	Click the New Object button on the Workbench toolbar.
C	In the Workbench left pane:

	Either	Or
	Double click Scripts .	a. Right-click Scripts . b. Select New on the Popup menu.
D	a. In the Workbench right pane. a. Right-click anywhere. b. Select New on the Popup menu.	
E	Press Ctrl+N on the keyboard.	

A Create Script dialog box opens.

Do the following.



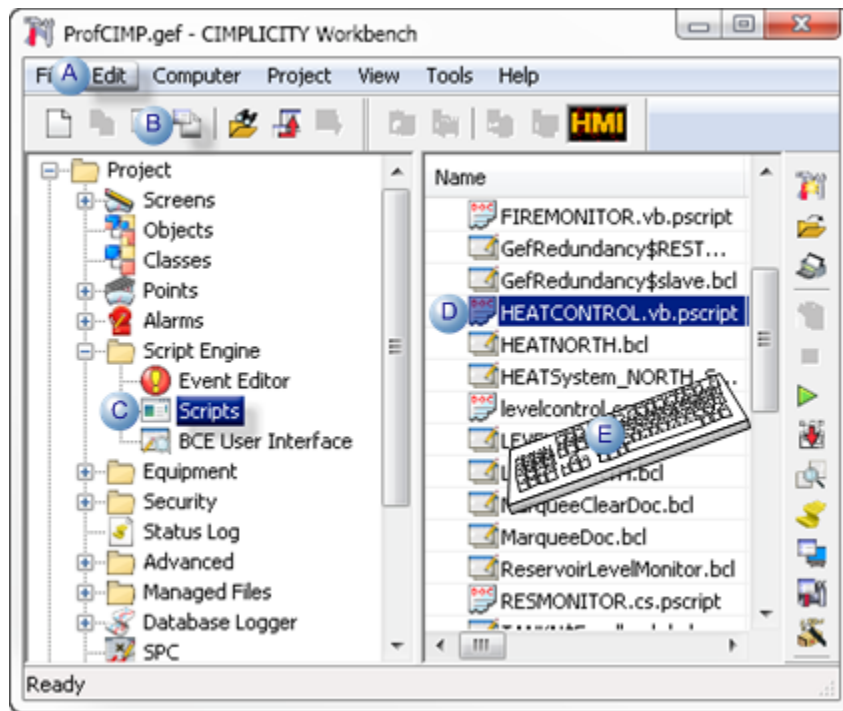
A	File Name	Enter a unique name to identify the script.	
B	Check one of the following.		
	Basic Script	Opens	CIMPLICITY Program Editor.
		Script file created	*.bcl
	C#	Opens	CimScriptIDE window.
		Script file created	*.cs.pscript
	Visual Basic.NET	Opens	CimScriptIDE window.
		Script file created	*.vb.pscript
C	Click one of the buttons.		
	Create	a. Creates the script. b. Opens the script editor window for the selected script type.	

Cancel	Closes the Create Script dialog box without creating a script.
--------	--

3. Right-click **Scripts**.
4. Select New on the Popup menu.
5. Right-click anywhere.
6. Select New on the Popup menu.
7. Creates the script.
8. Opens the script editor window for the selected script type.

1.2. Open an Existing C# or VB .NET Script

1. Select **Project>Script Engine>Scripts** in the Workbench left pane.
2. Select a *.cs.pscript or *.vb.pscript file in the Workbench right pane.
3. Do one of the following.



A	Click Edit>Properties on the Workbench menu bar.
B	Click the Properties button on the Workbench toolbar.
C	In the Workbench left pane: <ol style="list-style-type: none"> a. Right-click Scripts. b. Select Properties on the Popup menu.

D	In the Workbench right pane:	
	Either	Or
	Double click a script.	<ul style="list-style-type: none"> a. Right-click a script. b. Select Properties on the Popup menu.
E	Press Alt+Enter on the keyboard.	

4. Right-click **Scripts**.
5. Select Properties on the Popup menu.
6. Right-click a script.
7. Select Properties on the Popup menu.

2. CimScriptIDE Editor: Overview

2. CimScriptIDE Editor: Overview

The CimScriptIDE editor:

- Supports and facilitates scripting in both `C#` and `VB .NET`.
- Includes features from the CIMPLICITY Program Editor that are familiar to CIMPLICITY users.
- Provides features that are designed specifically for C# and VB.Net scripting.



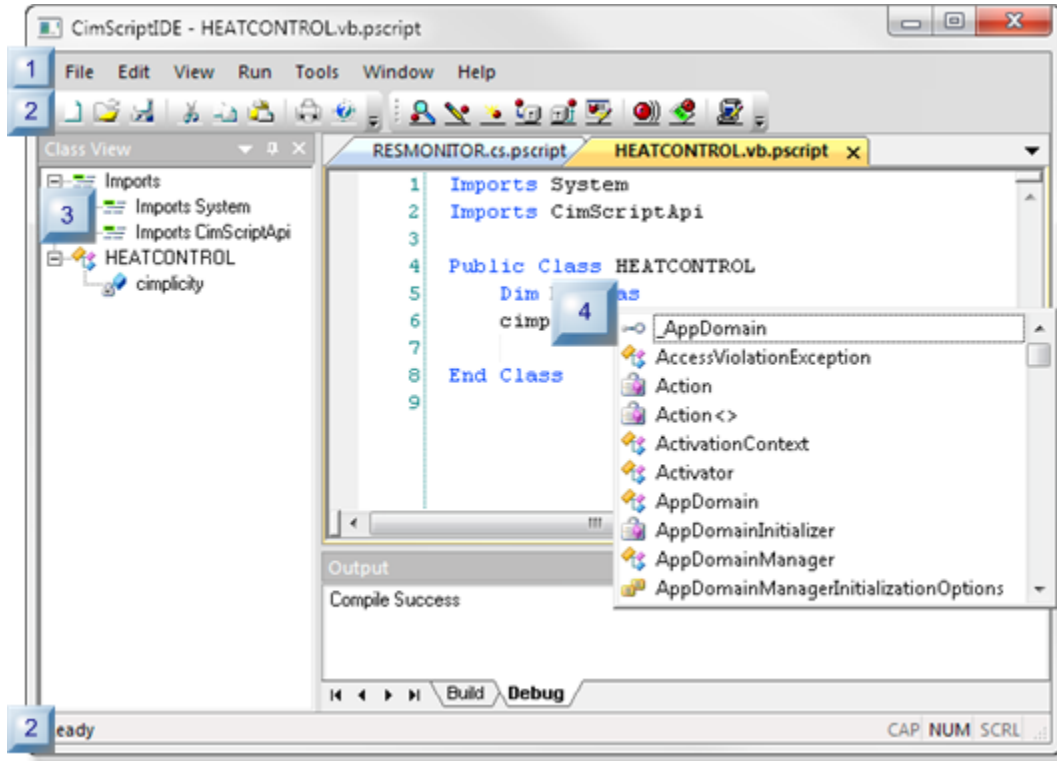
Important:

CimScriptIDE editor uses `.NET 4.5`, which was a required installation when CIMPLICITY v9.0 was installed. However, the CimScriptIDE editor does not recognize certain keywords that are new in `.NET 4.5` and will display an error message when one is not recognized. However, you can still compile and run scripts that contain the unrecognized keywords.

Example

CimScriptIDE editor does not recognize these keywords.

- `async`
- `await`



1. 2.4. CimScriptIDE Editor: Right-Pane (on page 264)
2. 2.3. CimScriptIDE Editor: Class View Pane (on page 263)
3. 2.2. CimScriptIDE Editor: Toolbars and Status Bar (on page 262)
4. 2.2. CimScriptIDE Editor: Toolbars and Status Bar (on page 262)
5. 2.1. CimScriptIDE Editor: Menus (on page 258)

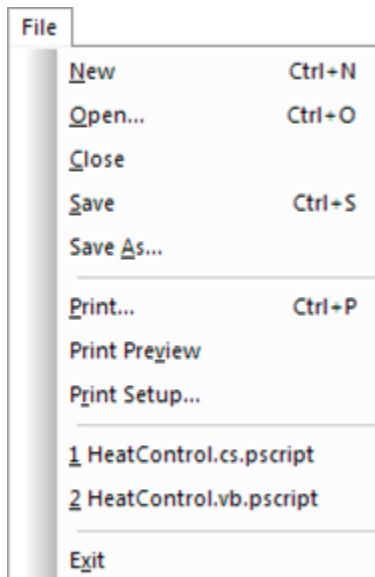
2.1 (on page 258)	CimScriptIDE Editor: Menus.
2.2 (on page 262)	CimScriptIDE Editor: Toolbars and status bar.
2.3 (on page 263)	CimScriptIDE Editor: Classes pane.
2.4 (on page 264)	CimScriptIDE Editor: Right-pane.

2.1. CimScriptIDE Editor: Menus

Menus in the CimScriptIDE editor are as follows.

- File menu.
- Edit menu.
- View menu.
- Run menu.
- Tools menu.
- Window menu.
- Help menu.

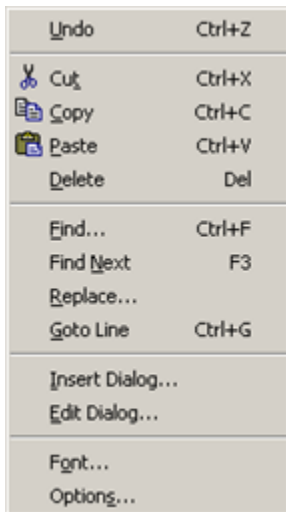
File Menu



New	Creates a new document for the Program Editor.
Open	Opens an existing document for the Program editor.
Close	Closes the script.
Save	Saves the active document.
Save As	Save the script with a different name.
Print	Prints the active document

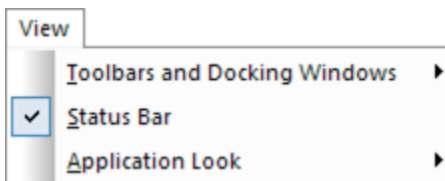
Print Pre-view	Displays the active document as it will be printed
Print Setup	Opens the Setup dialog box for the default printer.
Recent Files	Displays the list of most recently accessed files.
Exit	Exits the Program Editor.

Edit Menu




Undo	Undoes actions, beginning with the last action performed.
Redo	Redoes the actions that have been undone, beginning with the last undo.
Cut	Cuts the selection and puts it on the Clipboard.
Copy	Copies the selection and puts it on the Clipboard
Paste	Inserts Clipboard contents.

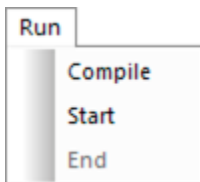
View Menu



Toolbars and Docking Windows	Displays the list of available toolbars. You can toggle the display of each toolbar.
------------------------------	--

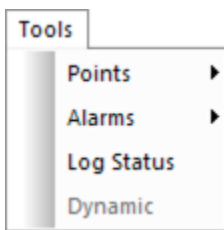
		Standard	Displays the Standard toolbar.
		Tools	Displays the Tools toolbar.
		Class View	Displays the CimScriptIDE Editor left-pane.
		Output	Displays the bottom right pane
		Customize	Opens the Customize dialog box.
Status Bar	Toggles the Status Bar at the bottom of the CimScriptIDE Editor.		

Run Menu

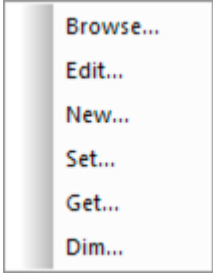
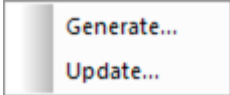


Compile	Compiles the script.
Start	Runs the program
End	Ends the running.

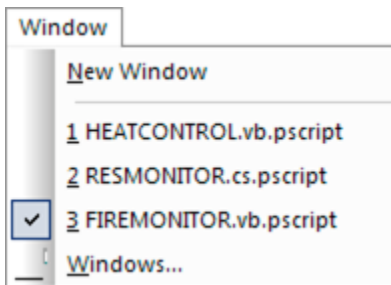
Tools Menu



Points	Displays a submenu that enables you to browse for points, edit a point, and create a new point. You can also use this menu item to include Setpoints, Getpoints and create local variables in the program.
--------	--

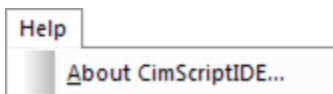
		Browse	Opens the Select a Point browser.
		Edit	Opens a selected point's Properties dialog box.
		New	Opens a New Point dialog box.
		Set	Opens a Set Point dialog box.
		Get	Opens a Get Point dialog box.
		Dim	Opens a Dimension Point Object dialog box.
Alarms		Displays a submenu that lets you generate or update alarms in the program.	
		Generate	Opens a Generate Alarm dialog box.
		Update	Opens an Update Alarm dialog box.
Log Status	Opens a Log Status dialog box enabling you to generate messages for the Status Log.		
Dynamic	Toggles Dynamic Configuration of points, alarm, etc. Note: When the project is running, dynamic is enabled for users who have been assigned the Dynamic Configuration privilege.		

Window Menu



New Window	Opens a new window.
Open Windows	Displays a list of open windows.
Windows	Opens a Windows dialog box.

Help Menu



About Cim-ScriptIDE	Opens an About CimScriptIDE message box with details about the distribution number and installed service upgrades.
---------------------	--

2.2. CimScriptIDE Editor: Toolbars and Status Bar

The CimScriptIDE Editor contains the following toolbars.

- CimScriptIDE Editor: Toolbars.
- CimScriptIDE Editor: Status bar.

CimScriptIDE Editor: Toolbars

The CimScriptIDE editor has the following toolbars.

- Standard
- Tools

Standard Toolbar



A	New	Create a new document.
B	Open	Open an existing document
C	Save	Save the active document
D	Cut	Cut the selection and put it on the Clipboard
E	Copy	Copy the selection and put it on the Clipboard
F	Paste	Insert Clipboard contents
G	Print	Print the active document
H	About	Display program information, version number, and copy-right

Tools Toolbar

Buttons on the Tools toolbar open the following browser and dialog boxes.



A	Browse Point	Select a Point browser.
B	Edit Point	Point Properties dialog box for a selected point.
C	New Point	New Point dialog box.
D	Get Point	Get Point dialog box.
E	Set Point	Set Point dialog box.
F	Dim Point	Dimension Point Object dialog box.
G	Gen Alarm	Generate an Alarm dialog box.
H	Update Alarm	Update Alarm dialog box.
I	Log Status	Log Status dialog box.

CimScriptIDE Editor: Status Bar

The CimScriptIDE editor status bar displays the following.



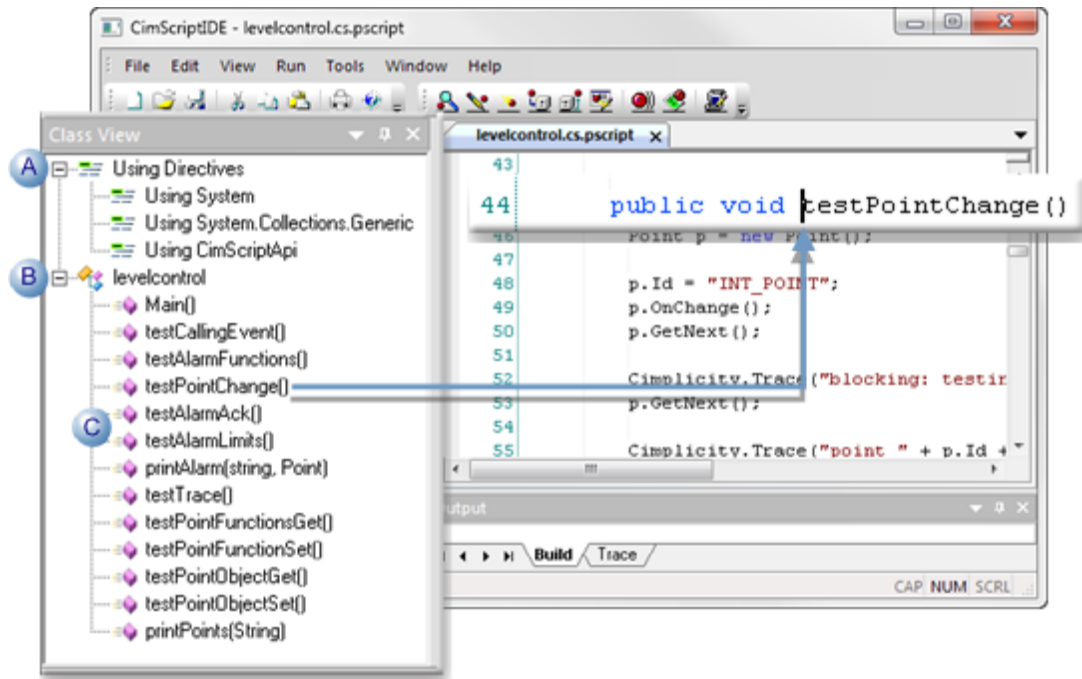
A	Displays status messages or tool tips when the mouse hovers over selected items, e.g. Ready or Copy the selection and put it on the clipboard.
B	Reports if the following keys are on or off. <ul style="list-style-type: none"> • CAP • NUM • SCRL


2.3. CimScriptIDE Editor: Class View Pane

The CimScriptIDE editor Class View pane enables you to easily

- Scan a script's imports, class nodes, function, properties, constants and class variables.
- Move the cursor to any selection by double-clicking the instance in the tree.

Note: Tree items can be expanded and collapsed.

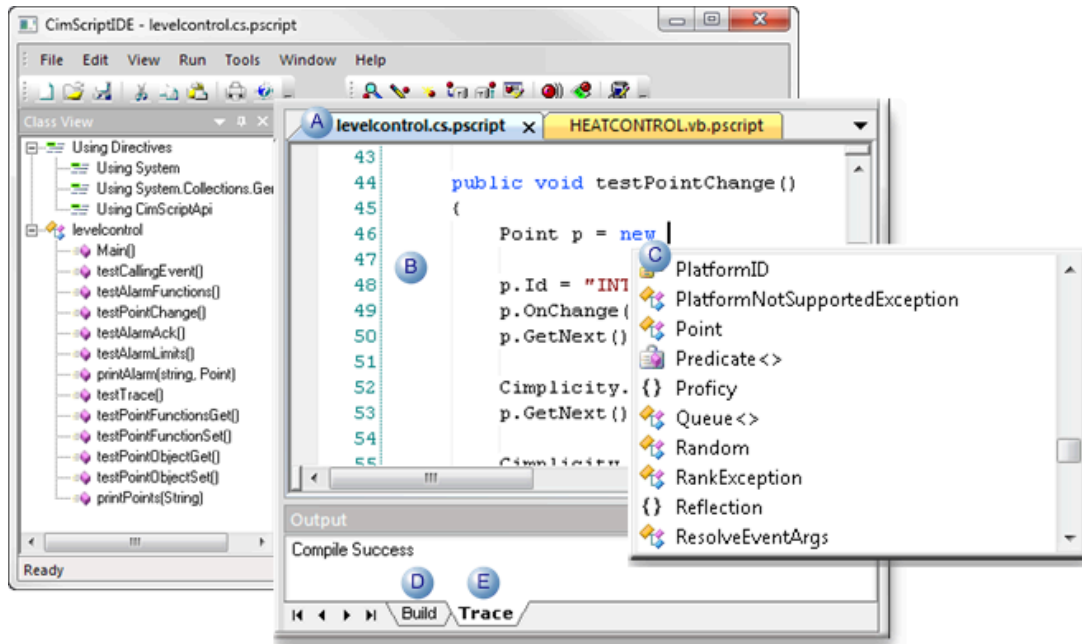


A	Based on the scripting type, a <code>Using</code> or <code>Imports</code> node can be expanded to list each entry in the script.	
	Script Type	Node
	C#	<code>Using</code>
	VB .NET	<code>Imports</code>
B	Class node	
	<div style="border: 1px solid #add8e6; border-radius: 10px; padding: 10px;"> <p> Note: This name must be the same as the script filename. If it is changed, make sure the filename is changed.</p> </div>	
C	Functions, nested classes, constants and class variables that are included in the class.	

2.4. CimScriptIDE Editor: Right-Pane

The CimScriptIDE editor right-pane provides a robust environment for creating and editing C Sharp and/or VB .NET scripts.

Features include the following.



A	Multiple scripts, which can be open at the same time, are identified by tabs at the top of the right-pane. The script for the selected tab (identified by an x) displays for editing.
B	The scripting area includes numbered lines.
C	As code is being written a CodeComplete Popup lists keywords, variables and members (methods, properties, and events) that can be used based on what was just written. Any item can be selected and automatically inserted.
D	A build tab displays compile errors.
E	A Trace tab traces messages from the script when the script is run.



Important:

The CimScriptIDE editor does not debug scripts; however, scripts written in the CimScriptIDE editor can be debugged live using Visual Studio.

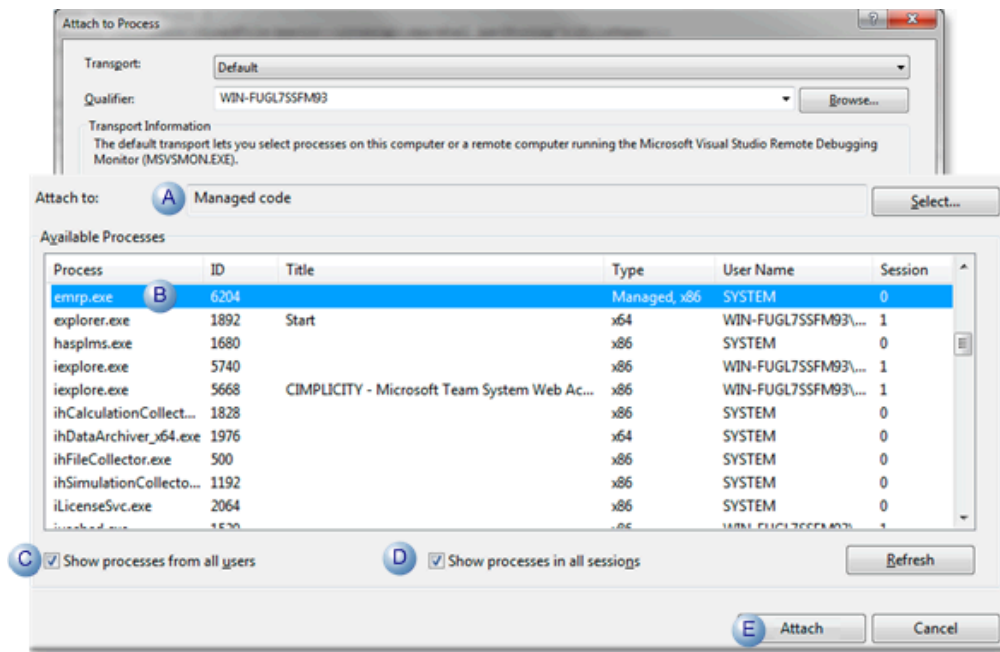
3. Technical Reference: CimScriptIDE Editor

3. Technical Reference: CimScriptIDE Editor

3.1 (on page 266)	CimScriptIDE debugging in Visual Studio.
3.2 (on page 267)	Attach Additional .NET Assembly references.

3.1. CimScriptIDE Debugging in Visual Studio

1. Create a CIMPLICITY event that will trigger the script.
2. Make sure the CIMPLICITY router is running.
3. Open Microsoft Visual Studio as an Administrator.
4. Click File>Open>File on the Visual Studio menu bar.
5. Find the script in the location you had saved it when you were in the CimScriptIDE editor.
6. Open the script.
7. Set the break points and trace points.
8. Select one of the following on the Visual Studio menu bar.
 - Tools>Attach to Process.
 - Debug>Attach to Process.
9. Do the following.



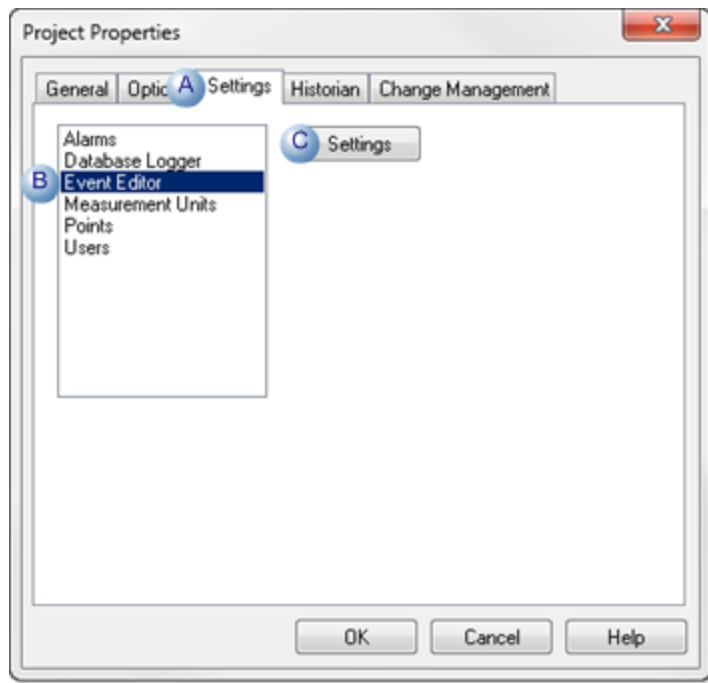
	Feature	Action
A	Attach to field.	Select Managed code.
B	Process list.	Select EMRP.exe.
C	Show processes from all users checkbox.	(Optional) Check
D	Show processes in all sessions checkbox.	(Optional) Check
E	Attach button.	Click.

The script will be triggered for debugging.

3.2. Attach Additional .NET Assembly References

CIMPLICITY provides default .NET assembly references for C# and VB .NET; additional references can be added or removed in the (Event Editor) Setup dialog box.

1. Open the Project Properties dialog box.
2. Do the following.

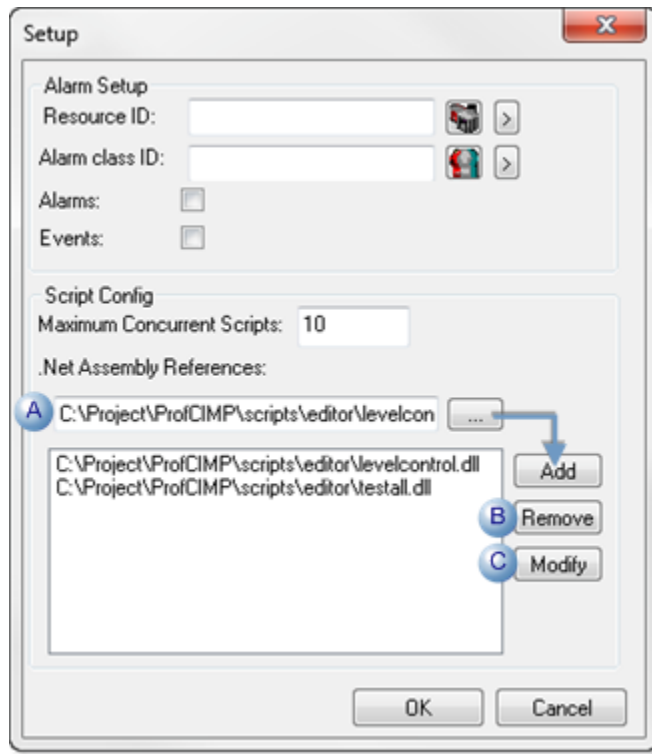


A	Select the Settings tab.
---	--------------------------

B	Select Event Editor.
C	Click Settings.

A Setup dialog box opens.

3. Do any of the following.



A	Add a .Net Assembly reference.	<p>a. Click the Open button to the right of the .Net Assembly References field.</p> <p>b. An Open dialog box opens.</p> <p>c. Select the .dll file that should be added to the list.</p> <p>d. Click Add.</p> <p>Result: The selected file is listed as one of the .Net Assembly references.</p>
B	Remove a reference.	Select the file to be removed; click Remove.
C	Modify the list.	Click Modify. The .Net Assembly Reference field is cleared.

4. Click the Open button to the right of the .Net Assembly References field.

5. An Open dialog box opens.

6. Select the .dll file that should be added to the list.
7. Click Add.

The selected file is listed as one of the .Net Assembly references.

Chapter 4. Basic Control Engine Language Reference

Using the Basic Control Engine Language Reference

The Basic Control Engine Language Reference documentation is organized like a dictionary containing an entry for each language element. The language elements are categorized as follows:

Category	Description
data type	Any of the support data types, such as Integer , String , and so on.
function	Language element that takes zero or more parameters, performs an action, and returns a value
keyword	Language element that doesn't fit into any of the other categories
operator	Language elements that cause an evaluation to be performed either on one or two operands
state- ment	Language element that takes zero or more parameters and performs an action.
topic	Describes information about a topic rather than a language element

Each entry in the Basic Control Engine Language Reference documentation contains the following headings:

Head- ing	Description
Syn- tax	The syntax of the language element. The conventions used in describing the syntax are described in Chapter 1 of the Basic Control Engine Language Reference documentation.
De- scrip- tion	Contains a one-line description of that language element.
Com- ments	Contains any other important information about that language keyword.
Exam- ple	Contains an example of that language keyword in use. An example is provided for every language keyword.
See Also	Contains a list of other entries in the Reference section that relate either directly or indirectly to that language element.

Scripting Language Reference

Click a cell entry to display the first topic in the Basic Control Engine Language Reference section.

Double-click Locate on the Help toolbar to locate the topic in the Table of Contents.

Basic Control Engine Language Reference

Intro (on page 272)
Symbols (on page 290)

A (on page 307)	G (on page 534)	N (on page 618)	T (on page 750)
B (on page 337)	H (on page 546)	O (on page 632)	U (on page 764)
C (on page 353)	I (on page 552)	P (on page 656)	V (on page 769)
D (on page 390)	K (on page 578)	Q (on page 676)	W (on page 778)
E (on page 460)	L (on page 580)	R (on page 677)	X (on page 795)
F (on page 510)	M (on page 600)	S (on page 697)	Y (on page 797)

CIMPLICITY Extensions to Basic (on page 797)
--

[CIMPLICITY Program Editor \(on page 110\)](#)

Object Model

CIMPLICITY Configuration
CIMPLICITY Visual Basic Extensions for CimBasic
CimLangMapper
CimEdit / CimView
CIMPLICITY Historical Data Connector
CIMPLICITY Historical Alarm Viewer
CIMPLICITY Safe Array
CIMPLICITY XY Plot
Tracker Agents
TADB
CIMPLICITY Solve Engine Interface
CIMPLICITY XML Translator

About the Basic Control Syntax

This section contains a complete, alphabetical listing of all keywords in the Basic Control Engine script language. When syntax is described, the following notations are used:

Notation	Description
While...Wend	Elements belonging to the Basic Control Engine script language, referred to in this manual as keywords, appear in the typeface shown to the left.
variable	Items that are to be replaced with information that you supply appear in italics. The type of replacement is indicated in the following description.
text\$	The presence of a type-declaration character following a parameter signifies that the parameter must be a variable of that type or an expression that evaluates to that type. If a parameter does not appear with a type-declaration character, then its type is described in the text.

Notation	Description	
[parameter]	Square brackets indicate that the enclosed items are optional. In Basic Control Engine script language, you cannot end a statement with a comma, even if the parameters are optional:	
	<code>MsgBox "Hello", , "Message" ' </code>	<code><--OK</code>
	<code>MsgBox "Hello", , ' </code>	<code><-- Not valid</code>
{Input Binary}	Braces indicate that you must choose one of the enclosed items, which are separated by a vertical bar.	
...	Ellipses indicate that the preceding expression can be repeated any number of times.	

Language Elements by Category

Language Elements By Category

The following subsections list Basic Control Engine language elements by category.

Arrays
Clipboard
Comments
Comparison operators
Controlling other programs
Controlling program flow
Controlling the operating environment
Conversion
Data types
Database
Date/time
DDE
Error handling
File I/O

File system
Financial
Getting information from Basic Control Engine
INI Files
Logical/binary operators
Math
Miscellaneous
Numeric operators
Objects
Parsing
Predefined dialogs
Printing
Procedures
String operators
Strings
User Dialogs
Variables and constants
Variants

Arrays

ArrayDims	Return the number of dimensions of an array
ArraySort	Sort an array
Erase	Erase the elements in one or more arrays
LBound	Return the lower bound of a given array dimension
Option Base	Change the default lower bound for array declarations

ReDim	Re-establish the dimensions of an array
UBound	Return the upper bound of a dimension of an array

Clipboard

Clipboard\$ (function)	Return the content of the clipboard as a string
Clipboard\$ (statement)	Set the content of the clipboard
Clipboard.Clear	Clear the clipboard
Clipboard.GetFormat	Get the type of data stored in the clipboard
Clipboard.GetText	Get text from the clipboard
Clipboard.SetText	Set the content of the clipboard to text

Comments

'	Comment to end-of-line
REM	Add a comment

Comparison Operators

<	Less than
<=	Less than or equal to
<>	Not equal
=	Equal
>	Greater than
>=	Greater than or equal to

Controlling other Programs

AppActivate	Activate an application
-------------	-------------------------

AppClose	Close an application
AppFind	Return the full name of an application
AppGetActive\$	Return the name of the active application
AppGetPosition	Get the position and size of an application
AppGetState	Get the window state of an application
AppHide	Hide an application
AppList	Fill an array with a list of running applications
AppMaximize	Maximize an application
AppMinimize	Minimize an application
AppMove	Move an application
AppRestore	Restore an application
AppSetState	Set the state of an application's window
AppShow	Show an application
AppSize	Change the size of an application
AppType	Return the type of an application
SendKeys	Send keystrokes to another application
Shell	Execute another application

Controlling Program Flow

Call	Call a subroutine
Choose	Return a value at a given index
Do...Loop	Execute a group of statements repeatedly
DoEvents (function)	Yield control to other applications
DoEvents (statement)	Yield control to other applications

End	Stop execution of a script
Exit Do	Exit a Do loop
Exit For	Exit a For loop
For...Next	Repeat a block of statement a specified number of times
GoSub	Execute at a specific label, allowing control to return later
Goto	Execute at a specific label
If...Then...Else	Conditionally execute one or more statements
IIf	Return one of two values depending on a condition
Main	Define a subroutine where execution begins
Return	Continue execution after the most recent GoSub
Select...Case	Execute one of a series of statements
Sleep	Pause for a specified number of milliseconds
Stop	Suspend execution, returning to a debugger (if present)
Switch	Return one of a series of expressions depending on a condition
While...Wend	Repeat a group of statements while a condition is True

Controlling the Operating Environment

Command, Command\$	Return the command line
Environm Environ\$	Return a string from the environment

Conversion

Asc	Return the value of a character
CBool	Convert a value to a Boolean
CCur	Convert a value to Currency
CDate	Convert a value to a Date

CDbl	Convert a value to a Double
Chr, Chr\$	Convert a character value to a string
CLng	Convert a value to a Long
CSng	Convert a value to a Single
CStr	Convert a value to a String
CVar	Convert a value to a Variant
CVDate	Convert a value to a Date
CVErr	Convert a value to an error
Hex, Hex\$	Convert a number to a hexadecimal string
IsDate	Determine if an expression is convertible to a date
IsError	Determine if a variant contains a user-defined error value
IsNumeric	Determine if an expression is convertible to a number
Oct, Oct\$	Convert a number to an octal string
Str, Str\$	Convert a number to a string
Val	Convert a string to a number

Data Types

Boolean	Data type representing True or False values
Currency	Data type used to hold monetary values
Date	Data type used to hold dates and times
Double	Data type used to hold real number with 15-16 digits of precision
HWND	Data type used to hold windows
Integer	Data type used to hold whole numbers with 4 digits of precision

Long	Data type used to hold whole numbers with 10 digits of precision
Object	Data type used to hold OLE automation objects
Single	Data type used to hold real number with 7 digits of precision
String	Data type used to hold sequences of characters
VARIANT	Data type that holds a number, string, or OLE automation objects

Database

SQLBind	Specify where to place results with SQLRetrieve
SQLClose	Close a connection to a database
SQLERROR	Return error information when an SQL function fails
SQLExecQuery	Execute a query on a database
SQLGetSchema	Return information about the structure of a database
SQLOpen	Establishes a connection with a database
SQLRequest	Run a query on a database
SQLRetrieve	Retrieve all or part of a query
SQLRetrieveToFile	Retrieve all or part of a query, placing results in a file

Date/time

Date, Date\$ (functions)	Return the current date
Date, Date\$ (statements)	Change the system date
DateAdd	Add a number of date intervals to a date
DateDiff	Subtract a number of date intervals from a date
DatePart	Return a portion of a date
DateSerial	Assemble a date from date parts

DateValue	Convert a string to a date
Day	Return the day component of a date value
Hour	Return the hour part of a date value
Minute	Return the minute part of a date value
Month	Return the month part of a date value
Now	Return the date and time
Second	Return the seconds part of a date value
Time, Time\$ (functions)	Return the current system time
Time, Time\$ (statements)	Set the system time
Timer	Return the number of elapsed seconds since midnight
TimeSerial	Assemble a date/time value from time components
TimeValue	Convert a string to a date/time value
Weekday	Return the day of the week of a date value
Year	Return the year part of a date value

DDE

DDEExecute	Execute a command in another application
DDEInitiate	Initiate a DDE conversation with another application
DDEPoke	Set a value in another application
DDERequest, DDERequest\$	Return a value from another application
DDESend	Establish a DDE conversation, then sets a value in another application
DDETerminate	Terminate a conversation with another application
DDETerminateAll	Terminate all conversations
DDETimeOut	Set the timeout used for non-responding applications

Error Handling

Erl	Return the line with the error
Err (function)	Return the error that caused the current error trap
Err (statement)	Set the value of the error
Error	Simulate a trappable runtime error
Error, Error\$	Return the text of a given error
On Error	Trap an error
Resume	Continue execution after an error trap

File I/O

Close	Close one or more files
Eof	Determine if the end-of-file has been reached
FreeFile	Return the next available file number
Get	Read data from a random or binary file
Input#	Read data from a sequential file into variables
Input, Input\$	Read a specified number of bytes from a file
Line Input #	Read a line of text from a sequential file
Loc	Return the record position of the file pointer within a file
Lock	Lock a section of a file
Lof	Return the number of bytes in an open file
Open	Open a file for reading or writing
Print #	Print data to a file
Put	Write data to a binary or random file
Reset	Close all open files
Seek	Return the byte position of the file pointer within a file
Seek	Set the byte position of the file pointer which a file
UnLock	Unlock part of a file

Width#	Specify the line width for sequential files
Write #	Write data to a sequential file

File System

ChDir	Change the current directory
ChDrive	Change the current drive
CurDir, Cur-Dir\$	Return the current directory
Dir, Dir\$	Return files in a directory
DiskDrives	Fill an array with valid disk drive letters
DiskFree	Return the free space on a given disk drive
FileAttr	Return the mode in which a file is open
FileCopy	Copy a file
FileDateTime	Return the date and time when a file was last modified
FileDirs	Fill an array with a subdirectory list
FileExists	Determine if a file exists
FileLen	Return the length of a file in bytes
FileList	Fill an array with a list of files
FileParse\$	Return a portion of a filename
GetAttr	Return the attributes of a file
Kill	Delete files from disk
MkDir	Create a subdirectory
Name	Rename a file
RmDir	Remove a subdirectory
SetAttr	Change the attributes of a file

Financial

DDB	Return depreciation of an asset using double-declining balance method
Fv	Return the future value of an annuity
IPmt	Return the interest payment for a given period of an annuity
IRR	Return the internal rate of return for a series of payments and receipts
MIRR	Return the modified internal rate of return
NPer	Return the number of periods of an annuity
Npv	Return the net present value of an annuity
Pmt	Return the payment for an annuity
PPmt	Return the principal payment for a given period of an annuity
Pv	Return the present value of an annuity
Rate	Return the interest rate for each period of an annuity
Sln	Return the straight-line depreciation of an asset
SYD	Return the Sum of Years' Digits depreciation of an asset

Getting information from Basic Control Engine

Basic.Capability	Return capabilities of the platform
Basic.Eoln\$	Return the end-of-line character for the platform
Basic.FreeMemory	Return the available memory
Basic.HomeDir\$	Return the directory where Basic Control Engine is located
Basic.OS	Return the platform id
Basic.PathSeparator\$	Return the path separator character for the platform
Basic.Version\$	Return the version of Basic Control Engine

INI Files

ReadIni\$	Read a string from an INI file
ReadIniSection	Read all the item names from a given section of an INI file
WriteIni	Write a new value to an INI file

Logical/binary Operators

And	Logical or binary conjunction
Eqv	Logical or binary equivalence
Imp	Logical or binary implication
Not	Logical or binary negation
Or	Logical or binary disjunction
Xor	Logical or binary exclusion

Math

Abs	Return the absolute value of a number
Atn	Return the arc tangent of a number
Cos	Return the cosine of an angle
Exp	Return e raised to a given power
Fix	Return the integer part of a number
Int	Return the integer portion of a number
Log	Return the natural logarithm of a number
Random	Return a random number between two values
Randomize	Initialize the random number generator
Rnd	Generate a random number between 0 and 1
Sgn	Return the sign of a number

Sin	Return the sine of an angle
Sqr	Return the square root of a number
Tan	Return the tangent of an angle

Miscellaneous

()	Force parts of an expression to be evaluated before others
_	Line continuation
Beep	Make a sound
Inline	Allow execution or interpretation of a block of text

Numeric Operators

*	Multiply
+	Add
-	Subtract
/	Divide
\	Integer divide
^	Power
Mod	Remainder

Objects

CreateObject	Instantiate an OLE automation object
GetObject	Return an OLE automation object from a file, or returns a previously instantiated OLE automation object
Is	Compare two object variables
Nothing	Value indicating no valid object

Parsing

Item\$	Return a range of items from a string
ItemCount	Return the number of items in a string
Line\$	Retrieve a line from a string
LineCount	Return the number of lines in a string
Word\$	Return a sequence of words from a string
Word- Count	Return the number of words in a string

Predefined Dialogs

AnswerBox	Display a dialog asking a question
AskBox\$	Display a dialog allowing the user to type a response
AskPassword\$	Display a dialog allowing the user to type a password
InputBox, InputBox\$	Display a dialog allowing the user to type a response
MsgBox (function)	Display a dialog containing a message and some buttons
MsgBox (state- ment)	Display a dialog containing a message and some buttons
OpenFilename\$	Display a dialog requesting a file to open
SaveFilename\$	Display a dialog requesting the name of a new file
SelectBox	Display a dialog allowing selection of an item from an array

Printing

Print	Print data to the screen
Spc	Print a number of spaces within a Print statement
Tab	Used with Print to print spaces up to a column position

Procedures

Declare	An external routine or a forward reference
Exit Function	Exit a function
Exit Sub	Exit a subroutine
Function...End	Create a user-defined function
Sub...End	Create a user-defined subroutine

String Operators

&	Concatenate two strings
Like	Compare a string against a pattern

Strings

Format, Format\$	Return a string formatted to a given specification
InStr	Return the position of one string within another
LCase, LCase\$	Convert a string to lower case
Left, Left\$	Return the left portion of a string
Len	Return the length of a string or the size of a data item
LSet	Left align a string or user-defined type within another
LTrim, LTrim\$	Remove leading spaces from a string
Mid, Mid\$	Return a substring from a string
Mid, Mid\$	Replace one part of a string with another
Option Compare	Change the default comparison between text and binary
Option CStrings	Allow interpretation of C-style escape sequences in strings

Right, Right\$	Return the right portion of a string
RSet	Right align a string within another
RTrim, RTrim\$	Remove trailing spaces from a string
Space, Space\$	Return a string os spaces
StrComp	Compare two strings
String, String\$	Return a string consisting of a repeated character
Trim, Trim\$	Trim leading and trailing spaces from a string
UCase, UCase\$	Return the upper case of a string

User Dialogs

Begin Dialog	Begin definition of a dialog template
CancelButton	Define a Cancel button within a dialog template
CheckBox	Define a combo box in a dialog template
ComboBox	Define a combo box in a dialog template
Dialog (function)	Invoke a user-dialog, returning which button was selected
Dialog (statement)	Invoke a user-dialog
DlgControlId	Return the id of a control in a dynamic dialog
DlgEnable	Determine if a control is enabled in a dynamic dialog
DlgEnable	Enable or disables a control in a dynamic dialog
DlgFocus	Return the control with the focus in a dynamic dialog
DlgFocus	Set focus to a control in a dynamic dialog
DlgListBoxArray	Set the content of a list box or combo box in a dynamic dialog
DlgListBoxArray	Set the content of a list box or combo box in a dynamic dialog
DlgSetPicture	Set the picture of a control in a dynamic dialog
DlgText (statement)	Set the content of a control in a dynamic dialog
DlgText\$ (function)	Return the content of a control in a dynamic dialog
DlgValue (function)	Return the value of a control in a dynamic dialog

DlgValue (statement)	Set the value of a control in a dynamic dialog
DlgVisible (function)	Determine if a control is visible in a dynamic dialog
DlgVisible (statement)	Set the visibility of a control in a dynamic dialog
DropListBox	Define a drop list box in a dialog template
GroupBox	Define a group box in a dialog template
ListBox	Add a list box to a dialog template
OKButton	Add an OK button to a dialog template
OptionButton	Add an option button to a dialog template
OptionGroup	Add an option group to a dialog template
Picture	Add a picture control to a dialog template
PictureButton	Add a picture button to a dialog template
PushButton	Add a push button to a dialog template
Text	Add a text control to a dialog template
TextBox	Add a text box to a dialog template

Variables and Constants

=	Assignment
Const	Define a constant
DefBool	Set the default data type to Boolean
DefCur	Set the default data type to Currency
DefDate	Set the default data type to Date
DefDbf	Set the default data type to Double
DefInt	Set the default data type to Integer
DefLng	Set the default data type to Long
DefObj	Set the default data type to Object
DefSng	Set the default data type to Single
DefStr	Set the default data type to String

DefVar	Set the default data type to Variant
Dim	Declare a local variable
Global	Declare variables for sharing between scripts
Let	Assign a value to a variable
Private	Declare variables accessible to all routines in a script
Public	Declare variables accessible to all routines in all scripts
Set	Assign an object variable
Type	Declare a user-defined data type

Variants

IsEmpty	Determine if a variant has been initialized
IsError	Determine if a variant contains a user-defined error
IsMissing	Determine if an optional parameter was specified
IsNull	Determine if a variant contains valid data
IsObject	Determine if an expression contains an object
VarType	Return the type of data stored in a variant

Symbols

Symbols

'	(keyword)
-	(operator)
#Const	(directive)
#If...Then...#Else	(directive)
&	(operator)
()	(keyword)

*	(operator)
.	(keyword)
/	(operator)
\	(operator)
^	(operator)
_	(keyword)
+	(operator)
<	(operator)
<=	(operator)
<>	(operator)
=	(operator)
=	(statement)
>	(operator)
>=	(operator)

' (keyword)

Syntax	' text
Description	Causes the compiler to skip all characters between this character and the end of the current line.
Comments	This is very useful for commenting your code to make it more readable.
Example	<pre>Sub Main() 'This whole line is treated as a comment. i\$ = "Strings" 'This is a valid assignment with a comment. This line will cause an error (the apostrophe is missing). End Sub</pre>
See Also	Rem (on page 690) (statement); Comments (on page 390) (topic).

- (operator)

Syntax 1	expression1 - expression2											
Syntax 2	- expression											
Description	Returns the difference between expression1 and expression2 or, in the second syntax, returns the negation of expression.											
Comments	<p>Syntax 1 The type of the result is the same as that of the most precise expression, with the following exceptions:</p> <table border="1"> <tr> <td>If one expression is</td> <td>and the other expression is</td> <td>then the type result is</td> </tr> <tr> <td>Long</td> <td>Single</td> <td>Double</td> </tr> <tr> <td>Boolean</td> <td>Boolean</td> <td>Integer</td> </tr> </table>			If one expression is	and the other expression is	then the type result is	Long	Single	Double	Boolean	Boolean	Integer
If one expression is	and the other expression is	then the type result is										
Long	Single	Double										
Boolean	Boolean	Integer										
	<p>A runtime error is generated if the result overflows its legal range. When either or both expressions are Variant, then the following additional rules apply:</p> <ul style="list-style-type: none"> • If expression1 is Null and expression2 is Boolean, then the result is Empty. Otherwise, if either expression is Null, then the result is Null. • Empty is treated as an Integer of value 0. • If the type of the result is an Integer variant that overflows, then the result is a Long variant. • If the type of the result is a Long, Single, or Date variant that overflows, then the result is a Double variant. 											
	<p>Syntax 2 If expression is numeric, then the type of the result is the same type as expression, with the following exception:</p> <ul style="list-style-type: none"> • If expression is Boolean, then the result is Integer. 											
	<p>In 2's compliment arithmetic, unary minus may result in an overflow with Integer and Long variables when the value of expression is the largest negative number representable for that data type. For example, the following generates an overflow error:</p> <pre> Sub Main() Dim a As Integer a = -32768 a = -a '<-- Generates overflow here. End Sub </pre>											

	When negating variants, overflow will never occur because the result will be automatically promoted: integers to longs and longs to doubles.
Example	<p>This example assigns values to two numeric variables and their difference to a third variable, then displays the result.</p> <pre> Sub Main() i% = 100 j# = 22.55 k# = i% - j# MsgBox "The difference is: " & k# End Sub </pre>
See Also	Operator Precedence (on page 648) (topic).

#Const (directive)

Syntax	<code>#Const constname = expression</code>
Description	Defines a preprocessor constant for use in the <code>#If...Then...#Else</code> statement.
Comments	Internally, all preprocessor constants are of type Variant. Thus, the expression parameter can be any type. Variables defined using <code>#Const</code> can only be used within the <code>#If...Then...#Else</code> statement and other <code>#Const</code> statements. Use the <code>Const</code> statement to define constants that can be used within your code.
Example	<pre> #Const SUBPLATFORM = "XP" #Const MANUFACTURER = "Windows" #Const TYPE = "Workstation" #Const PLATFORM = MANUFACTURER & " " & SUBPLATFORM & " " & TYPE </pre>

	<pre> Sub Main() #If PLATFORM = "Windows XP Workstation" Then MsgBox "Running under Windows XP Workstation" #End If End Sub </pre>
See Also	#If...Then...#Else (on page 294) (directive)

#If...Then...#Else (directive)

Syntax	<pre> #If expression Then [statements] [#ElseIf expression Then [statements]] [#Else [statements]] #End If </pre>
Description	Causes the compiler to include or exclude sections of code based on conditions.

Com- ments	The expression represents any valid BasicScript Boolean expression evaluating to TRUE or FALSE. The expression may consist of literals, operators, constants defined with #Const, and any of the following predefined constants:		
	Constant	Value	
	Win32	True if development environment is 32-bit Windows.	
	Empty	Empty	
	FALSE	False	
	NULL	Null	
	TRUE	True	
	The expression can use any of the following operators: +, -, *, /, \, ^, + (unary), - (unary), Mod, &, =, <>, >=, >, <=, <, And, Or, Xor, Imp, Eqv.		
	Following are results when an expression is evaluated.		
	Evaluates to a:		Result
	Numeric value	Non-zero	TRUE
		Zero	FALSE
	String not convertible to a number		Type mismatch error is generated.
	Null		Type mismatch error is generated.
	Text comparisons within expression are always case-insensitive, regardless of the Option Compare setting You can define your own constants using the #Const directive, and test for these constants within the expression parameter as shown below:		
	<pre> #Const VERSION = 2 Sub Main #If VERSION = 1 Then directory\$ = "\apps\widget" #ElseIf VERSION = 2 Then directory\$ = "\apps\widget32" #Else MsgBox "Unknown version." #End If End Sub </pre>		

Any constant not already defined evaluates to **Empty**. A common use of the `#If...Then...#Else` directive is to optionally include debugging statements in your code. The following example shows how debugging code can be conditionally included to check parameters to a function:

```

        #Const DEBUG = 1

        Sub ChangeFormat(NewFormat As Integer,StatusText As String)

#If DEBUG = 1 Then

    If NewFormat <> 1 And NewFormat <> 2 Then

        MsgBox "Parameter ""NewFormat"" is invalid."

        Exit Sub

    End If

    If Len(StatusText) > 78 Then

        MsgBox "Parameter ""StatusText"" is too long."

        Exit Sub

    End If

#End If

    Rem Change the format here...

        End Sub

```

Excluded sections are not compiled by BasicScript, allowing you to exclude sections of code that has errors or doesn't even represent valid BasicScript syntax. For example, the following code uses the `#If...Then...#Else` statement to include a multi-line comment:

```

        Sub Main

#If 0

    The following section of code displays

        a dialog box containing a message and an

        OK button.

#End If

    MsgBox "Hello, world."

        End Sub

```

In the above example, since the expression `#If 0` never evaluates to TRUE, the text between that and the matching `#End If` will never be compiled.

See	#Const (directive) (on page 293)
Also	

& (operator)

Syn- tax	expression1 & expression2
De- scrip- tion	Returns the concatenation of expression1 and expression2.
Com- ments	If both expressions are strings, then the type of the result is String . Otherwise, the type of the result is a String variant. When nonstring expressions are encountered, each expression is converted to a String variant. If both expressions are Null , then a Null variant is returned. If only one expression is Null , then it is treated as a zero-length string. Empty variants are also treated as zero-length strings. In many instances, the plus (+) operator can be used in place of & . The difference is that + attempts addition when used with at least one numeric expression, whereas & always concatenates.
Exam- ple	This example assigns a concatenated string to variable s\$ and a string to s2\$, then concatenates the two variables and displays the result in a dialog box. <pre> Sub Main() s\$ = "This string" & " is concatenated" s2\$ = " with the '&' operator." MsgBox s\$ & s2\$ End Sub </pre>
See Also	+ (on page 303) (operator) ; Operator Precedence (on page 648) (topic) .

() (keyword)

Syn- tax 1	... (expression) ...
Syn- tax 2	..., (parameter) ,...

De- scrip- tion	Forces parts of an expression to be evaluated before others or forces a parameter to be passed by value.
Com- ments	<p>Parentheses within Expressions Parentheses override the normal precedence order of the scripts operators, forcing a subexpression to be evaluated before other parts of the expression. For example, the use of parentheses in the following expressions causes different results: <code>i = 1 + 2 * 3</code> 'Assigns 7. <code>i = (1 + 2) * 3</code> 'Assigns 9. Use of parentheses can make your code easier to read, removing any ambiguity in complicated expressions.</p>
	<p>Parentheses Used in Parameter Passing Parentheses can also be used when passing parameters to functions or subroutines to force a given parameter to be passed by value, as shown below: <code>ShowForm i</code> 'Pass i by reference. <code>ShowForm (i)</code> 'Pass i by value. Enclosing parameters within parentheses can be misleading. For example, the following statement appears to be calling a function called ShowForm without assigning the result: <code>ShowForm(i)</code> The above statement actually calls a subroutine called ShowForm, passing it the variable <code>i</code> by value. It may be clearer to use the ByVal keyword in this case, which accomplishes the same thing: ShowForm ByVal i The result of an expression is always passed by value.</p>
Exam- ple	<p>This example uses parentheses to clarify an expression.</p> <pre> Sub Main() bill = False dave = True jim = True If (dave And bill) Or (jim And bill) Then MsgBox "The required parties for the meeting are here." Else MsgBox "Someone is late for the meeting!" End If End Sub </pre>
See Also	ByVal (on page 352) (keyword); Operator Precedence (on page 648) (topic).

* (operator)

Syntax	expression1 * expression2
--------	---------------------------

De- scrip- tion	Returns the product of expression1 and expression2.		
Com- ments	The result is the same type as the most precise expression, with the following exceptions:		
	If one expression is	and the other expression is	then the type the result is
	Single	Long	Double
	Boolean	Boolean	Integer
	Date	Date	Double
	<p>When the * operator is used with variants, the following additional rules apply:</p> <ul style="list-style-type: none"> • Empty is treated as 0. • If the type of the result is an Integer variant that overflows, then the result is automatically promoted to a Long variant. • If the type of the result is a Single, Long, or Date variant that overflows, then the result is automatically promoted to a Double variant. • If expression1 is Null and expression2 is Boolean, then the result is Empty. Otherwise, If either expression is Null, then the result is Null. 		
Exam- ple	<p>This example assigns values to two variables and their product to a third variable, then displays the product of s# * t#.</p> <pre> Sub Main() s# = 123.55 t# = 2.55 u# = s# * t# MsgBox s# & " * " & t# & " = " & s# * t# End Sub </pre>		
See Al- so	Operator Precedence (on page 648) (topic)		

. (keyword)

Syntax 1	object . property
-------------	-------------------

Syntax 2	structure.member
De- scrip- tion	Separates an object from a property or a structure from a structure member.
Exam- ples	This example uses the period to separate an object from a property. Sub Main() MsgBox "The clipboard text is: " & Clipboard.GetText() End Sub
	<p>This example uses the period to separate a structure from a member.</p> <pre> Type Rect left As Integer top As Integer right As Integer bottom As Integer End Type Sub Main() Dim r As Rect r. left = 10 r. righth = 12 MsgBox "r.left = " & r.left & ", r.right = " & r.right End Sub </pre>
See Al- so	Objects (on page 285) (topic).

/ (operator)

Syntax	expression1 / expression2		
De- scrip- tion	Returns the quotient of expression1 and expression2.		
Com- ments	The type of the result is Double , with the following exceptions:		
	If one expression is	and the other expression is	then the type the result is
	Integer	Integer	Single

	Single	Single	Single
	Boolean	Boolean	Single
	<p>A runtime error is generated if the result overflows its legal range. When either or both expressions is Variant, then the following additional rules apply:</p> <ul style="list-style-type: none"> • If expression1 is Null and expression2 is Boolean, then the result is Empty. Otherwise, if either expression is Null, then the result is Null. • Empty is treated as an Integer of value 0. • If both expressions are either Integer or Single variants and the result overflows, then the result is automatically promoted to a Double variant. 		
Example	<p>This example assigns values to two variables and their quotient to a third variable, then displays the result.</p> <pre> Sub Main() i% = 100 j# = 22.55 k# = i% / j# MsgBox "The quotient of i/j is: " & k# End Sub </pre>		
See Also	<p>\ (on page 301) (operator): Operator Precedence (on page 648) (topic)</p>		

\ (operator)

Syntax	expression1 \ expression2
Description	Returns the integer division of expression1 and expression2.
Comments	Before the integer division is performed, each expression is converted to the data type of the most precise expression. If the type of the expressions is either Single , Double , Date , or Currency , then each is rounded to Long . If either expression is a Variant , then the following additional rules apply:

	<ul style="list-style-type: none"> • If either expression is Null , then the result is Null . • Empty is treated as an Integer of value 0 .
Example	<p>This example assigns the quotient of two literals to a variable and displays the result.</p> <pre>Sub Main() s% = 100.99 \ 2.6 MsgBox "Integer division of 100.99\2.6 is: " & s% End Sub</pre>
See Also	/ (on page 300) (operator); Operator Precedence (on page 648) (Topic)

^ (operator)

Syntax	expression1 ^ expression2	
Description	Returns expression1 raised to the power specified in expression2.	
Comments	The following are special cases:	
	Special Case	Value
	<code>n^0</code>	1
	<code>0^-n</code>	Undefined
	<code>0^+n</code>	0
	<code>1^n</code>	1
	The type of the result is always Double , except with Boolean expressions, in which case the result is Boolean . Fractional and negative exponents are allowed. If either expression is a Variant containing NULL, then the result is NULL. It is important to note that raising a number to a negative exponent produces a fractional result.	
Example	<pre>Sub Main() s# = 2 ^ 5 'Returns 2 to the 5th power. r# = 16 ^ .5 'Returns the square root of 16. MsgBox "2 to the 5th power is: " & s#</pre>	

	<pre>MsgBox "The square root of 16 is: " & r# End Sub</pre>
See Also	Operator Precedence (on page 648) (topic).

_ (keyword)

Syntax	s\$ = "This is a very long line that I want to split " & _ "onto two lines"
Description	Line-continuation character, which allows you to split a single script onto more than one line.
Comments	The line-continuation character cannot be used within strings and must be preceded by white space (either a space or a tab). The line-continuation character can be followed by a comment, as shown below: <code>i = 5 + 6 & _ 'Continue on the next line. "Hello"</code>
Example	<pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() 'The line-continuation operator is useful when concatenating 'long strings. msg1 = "This line is a line of text that" & crlf & "extends beyond " _ & "the borders of the editor" & crlf & "so it is split into " _ & "multiple lines" 'It is also useful for separating and continuing long calculation lines. b# = .124 a# = .223 s# = (((Sin(b#) ^ 2) + (Cos(a#) ^ 2)) ^ .5) / _ (((Sin(a#) ^ 2) + (Cos(b#) ^ 2)) ^ .5) * 2.00 MsgBox msg1 & crlf & crlf & "The value of s# is: " & s# End Sub</pre>

+ (operator)

Syntax	expression1 + expression2
--------	----------------------------------

De- scrip- tion	Adds or concatenates two expressions.		
Com- ments	Addition operates differently depending on the type of the two expressions:		
	If one expression is	and the other expression is	then
	Numeric	Numeric	Perform a numeric add (see below).
	String	String	Concatenate, returning a string.
	Numeric	String	A runtime error is generated.
	Variant	String	Concatenate, returning a String variant.
	Variant	Numeric	Perform a variant add (see below).
	Empty variant	Empty variant	Return an Integer variant, value 0 .
	Empty variant	Boolean variant	Return an Integer variant (value 0 or -1)
	Empty variant	Any data type	Return the non- Empty expression unchanged.
	Null variant	Any data type	Return Null .
	Variant	Variant	If either is numeric, add; otherwise, concate- nate.
	When using + to concatenate two variants, the result depends on the types of each variant at runtime. You can remove any ambiguity by using the & operator. Numeric Add A numeric add is performed when both expressions are numeric (i.e., not variant or string). The result is the same type as the most precise expression, with the following exceptions:.		
	If one expression is	and the other expression is	then
	Single	Long	Double
	Boolean	Boolean	Integer
	A runtime error is generated if the result overflows its legal range Variant Add If both expressions are variants, or one expression is numeric and the other expression is Variant , then a variant add is performed. The rules for variant add are the same as those for normal numeric add, with the following exceptions:		

	<ul style="list-style-type: none"> • If the type of the result is an Integer variant that overflows, then the result is a Long variant. • If the type of the result is a Long, Single, or Date variant that overflows, then the result is a Double variant.
Example	<p>This example assigns string and numeric variable values and then uses the + operator to concatenate the strings and form the sums of numeric variables.</p> <pre> Sub Main() i\$ = "concatenate " + "strings!" j% = 95 + 5 'Addition of numeric literals k# = j% + j% 'Addition of numeric variable MsgBox "You can " + i\$ MsgBox "You can add literals or variables:" + Str(j%) + ", " + Str(k#) End Sub </pre>
See Also	<p>& (on page 297) (Operator); Operator Precedence (on page 648) (topic)</p>

< (operator)

See [Comparison Operators \(on page 275\)](#) (topic).

<= (operator)

See [Comparison Operators \(on page 275\)](#) (topic).

<> (operator)

See [Comparison Operators \(on page 275\)](#) (topic).

= (operator)

See [Comparison Operators \(on page 275\)](#) (topic).

= (statement)

Syn- tax	variable = expression
De- scrip- tion	Assigns the result of an expression to a variable.
Com- ments	When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This occurs when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error: Dim amount As Long Dim quantity As Integer amount = 400123 'Assign a value out of range for int. quantity = amount 'Attempt to assign to Integer. When performing an automatic data conversion, underflow is not an error.
	The assignment operator (=) cannot be used to assign objects. Use the Set statement instead.
Exam- ple	<pre>Sub Main() a\$ = "This is a string" b% = 100 c# = 1213.3443 MsgBox a\$ & ", " & b% & ", " & c# End Sub</pre>
See Also	Let (on page 586) (statement); Operator Precedence (on page 648) (topic); Set (on page 714) (statement); Expression Evaluation (on page 509) (topic).

> (operator)

See [Comparison Operators \(on page 275\)](#) (topic).

>= (operator)

See [Comparison Operators \(on page 275\)](#) (topic).

A

A

Abs (function)
And (operator)
AnswerBox (function)
Any (data type)
AppActivate (statement)
AppClose (statement)
AppFind, AppFind\$ (functions)
AppGetActive\$ (function)
AppGetPosition (statement)
AppGetState (function)
AppHide (statement)
AppList (statement)
AppMaximize (statement)
AppMinimize (statement)
AppMove (statement)
AppRestore (statement)
AppSetState (statement)
AppShow (statement)
AppSize (statement)
AppType (function)
ArrayDims (function)

Arrays (topic)
ArraySort (statement)
Asc, AscB, AscW (functions)
AskBox, AskBox\$ (functions)
AskPassword, AskPassword\$ (functions)
Atn (function)

Abs (function)

Syntax	Abs (expression)
Description	Returns the absolute value of expression.
Comments	<p>If expression is Null, then Null is returned. Empty is treated as 0. The type of the result is the same as that of expression, with the following exceptions:</p> <ul style="list-style-type: none"> If expression is an Integer that overflows its legal range, then the result is returned as a Long. This only occurs with the largest negative Integer: <pre>Dim a As Variant Dim i As Integer i = -32768 a = Abs(i) 'Result is a Long. i = Abs(i) 'Overflow!</pre> <ul style="list-style-type: none"> If expression is a Long that overflows its legal range, then the result is returned as a Double. This only occurs with the largest negative Long: <pre>Dim a As Variant Dim l As Long l = -2147483648 a = Abs(l) 'Result is a Double. l = Abs(l) 'Overflow!</pre>

	<ul style="list-style-type: none"> If expression is a Currency value that overflows its legal range, an overflow error is generated.
Example	<p>This example assigns absolute values to variables of four types and displays the result.</p> <pre> Sub Main() s1% = Abs(-10.55) s2& = Abs(-10.55) s3! = Abs(-10.55) s4# = Abs(-10.55) MsgBox "The absolute values are: " & s1% & ", " & s2& & ", " & s3! & ", " & s4# End Sub </pre>
See Also	Sgn (on page 716) (function).

And (operator)

Syntax	expression1 And expression2		
Description	Performs a logical or binary conjunction on two expressions.		
Comments	If both expressions are either Boolean , Boolean variants, or Null variants, then a logical conjunction is performed as follows:		
	If the first expression is	and the second expression is	then the result is
	True	True	True
	True	False	False
	True	Null	Null
	False	True	False
	False	False	False
	False	Null	Null
	Null	True	Null
	Null	False	False

	Null	Null
<p>Binary Conjunction If the two expressions are Integer, then a binary conjunction is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long, and a binary conjunction is then performed, returning a Long result. Binary conjunction forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:</p>		

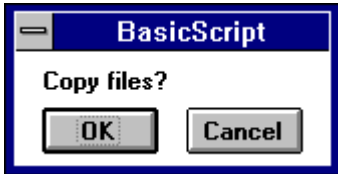
	1	And	1	=	1	Example:
	0	And	1	=	0	5 00001001
	1	And	0	=	0	6 00001010
	0	And	0	=	0	And 00001000


Exam- ple	<pre> Sub Main() n1 = 1001 n2 = 1000 b1 = True b2 = False 'This example performs a numeric bitwise And operation and stores 'the result in N3. n3 = n1 And n2 'This example performs a logical And comparing b1 and b2 and displays 'the result. If b1 And b2 Then MsgBox "b1 And b2 are True; n3 is: " & n3 Else MsgBox "b1 And b2 are False; n3 is: " & n3 End If End Sub </pre>
----------------------	---

See Al- so	Operator Precedence (on page 648) (topic); Or (on page 654) (operator); Xor (on page 795) (operator); Eqv (on page 489) ;(operator); (operator) (on page 557).
---------------	--

AnswerBox (function)

Syn- tax	<code>AnswerBox(prompt [,button1] [,button2] [,button3]!!!!)</code>
-------------	---

De- scrip- tion	Displays a dialog box prompting the user for a response and returns an Integer indicating which button was clicked (1 for the first button, 2 for the second, and so on).	
Com- ments	The AnswerBox function takes the following parameters:	
	Parameter	Description
	Prompt	Text to be displayed above the text box. The prompt parameter can be any expression convertible to a String .
		The Basic Control Engine script resizes the dialog box to hold the entire contents of prompt, up to a maximum width of 5/8 of the width of the screen and a maximum height of 5/8 of the height of the screen. It also word-wraps any lines too long to fit within the dialog box and truncates all lines beyond the maximum number of lines that fit in the dialog box.
		You can insert a carriage-return/line-feed character in a string to cause a line break in your message.
		A runtime error is generated if this parameter is Null .
	Button1	Text for the first button. If omitted, then "OK" and "Cancel" are used. A runtime error is generated if this parameter is Null .
	Button2	Text for the second button. A runtime error is generated if this parameter is Null .
	Button3	Text for the third button. A runtime error is generated if this parameter is Null .
	<p>The width of each button is determined by the width of the widest button. The AnswerBox function returns 0 if the user selects Cancel. <code>R% = AnswerBox("Copy files?")</code></p>  <p><code>R% = AnswerBox("Copy files?","Save","Restore","Cancel")</code></p>	

	
Example	<p>This example displays a dialog box containing three buttons. It displays an additional message based on which of the three buttons is selected.</p> <pre> Sub Main() r% = AnswerBox("Temporary File Operation?", "Save", "Remove", "Cancel") Select Case r% Case 1 MsgBox "Files will be saved." Case 2 MsgBox "Files will be removed." Case Else MsgBox "Operation canceled." End Select End Sub </pre>
See Also	<p>MsgBox (on page 617) (statement); AskBox\$ (on page 333) (function); AskPassword\$ (on page 335) (function); InputBox, InputBox\$ (on page 562) (functions); OpenFilename\$ (on page 646) (function); SaveFilename\$ (on page 699) (function); SelectBox (on page 709) (function).</p>
Notes	<p>AnswerBox displays all text in its dialog box in 8-point MS Sans Serif.</p>

Any (data type)

De- scrip- tion	<p>Used with the Declare statement to indicate that type checking is not to be performed with a given argument.</p>
Com- ments	<p>Given the following declaration:</p> <pre> Declare Sub Foo Lib "FOO.DLL" (a As Any) </pre> <p>The following calls are valid:</p> <pre> Foo 10 Foo "Hello, world." </pre>

Example	<p>The following example calls the FindWindow to determine if Program Manager is running. This example uses the Any keyword to pass a NULL pointer, which is accepted by the FindWindow function.</p> <pre data-bbox="305 352 1386 1121"> Declare Function FindWindow16 Lib "user" Alias "FindWindow" (ByVal Class _ As Any,ByVal Title As Any) As Integer Declare Function FindWindow32 Lib "user32" Alias "FindWindowA" (ByVal Class _ As Any,ByVal Title As Any) As Long Sub Main() Dim hWnd As Variant If Basic.Os = ebWin16 Then hWnd = FindWindow 16("PROGMAN",0&) ElseIf Basic.Os = ebWin32 Then hWnd = FindWindow32("PROGMAN",0&) Else hWnd = 0 End If If hWnd <> 0 Then MsgBox "Program manager is running, window handle is " & hWnd End If End Sub </pre>
See Also	<p>Declare (on page 412) (statement).</p>

AppActivate (statement)

Syntax	AppActivate name\$ taskID	
Description	Activates an application given its name or task ID.	
Comments	The AppActivate statement takes the following parameters:	
	Parameter	Description

	Name\$	String containing the name of the application to be activated.
	TaskID	Number specifying the task ID of the application to be activated. Acceptable task IDs are returned by the Shell function
	When activating applications using the task ID, it is important to declare the variable used to hold the task ID as a VARIANT . The type of the ID depends on the platform on which The Basic Control Engine script is running.	
Example 1	<p>This example activates Program Manager.</p> <pre>Sub Main() AppActivate "Program Manager" End Sub</pre>	
Example 2	<p>This example runs another application, activates it, and maximizes it.</p> <pre>Sub Main() Dim id as variant id = Shell("notepad.exe") 'Run Notepad minimized. AppActivate id 'Now activate Notepad. AppMaximize End Sub</pre>	
See Also	Shell (on page 717) (function); SendKeys (on page 711) (statement); WinActivate (on page 782) (statement).	
Notes	<ul style="list-style-type: none"> • The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. • Minimized applications are not restored before activation. Thus, activating a minimized DOS application will not restore it; rather, it will highlight its icon. • A runtime error results if the window being activated is not enabled, as is the case if that application is currently displaying a modal dialog box. 	

AppClose (statement)

Syntax	AppClose [name\$]
--------	--------------------------

De- scrip- tion	Closes the named application.
Com- ments	The name\$ parameter is a String containing the name of the application. If the name\$ parameter is absent, then the AppClose statement closes the active application.
Exam- ple	<p>This example activates Excel, then closes it.</p> <pre> Sub Main() If AppFind\$("Microsoft Excel") = "" Then 'Make sure Excel is there. MsgBox "Excel is not running." Exit Sub End If AppActivate "Microsoft Excel" 'Activate it (unnecessary). AppClose "Microsoft Excel" 'Close it. End Sub </pre>
See Also	AppMaximize (on page 320) (statement); AppMinimize (on page 321) (statement); AppRestore (on page 323) (statement); AppMove (on page 322) (statement); AppSize (on page 326) (statement).
Notes	A runtime error results if the application being closed is not enabled, as is the case if that application is currently displaying a modal dialog box. The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used.

AppFind, AppFind\$ (functions)

Syn- tax	AppFind[\$] (title taskID)
De- scrip- tion	Returns a String containing the full name of the application matching either title or taskID.
Com- ments	The title parameter specifies the title of the application to find. If there is no exact match, BasicScript will find an application whose title begins with title. Alternatively, you can specify the ID of the task as returned by the <code>Shell</code> function. The <code>AppFind\$</code> functions returns a String , where-

	<p>as the <code>AppFind</code> function returns a String variant. If the specified application cannot be found, then <code>AppFind\$</code> returns a zero-length string and <code>AppFind</code> returns Empty. Using <code>AppFind</code> allows you detect failure when attempting to find an application with no caption (i.e., Empty is returned instead of a zero-length String). <code>AppFind\$</code> is generally used to determine whether a given application is running. The following expression returns True if Microsoft Word is running:</p> <pre>AppFind\$("Microsoft Word")</pre>
Example	<pre>'This example checks to see whether Excel is running before 'activating it. Sub Main() If AppFind\$("Microsoft Excel") <> "" Then AppActivate "Microsoft Excel" Else MsgBox "Excel is not running." End If End Sub</pre>
Notes	<p>This function returns a String containing the exact text appearing in the title bar of the active application's main window.</p>

AppGetActive\$ (function)

Syntax	AppGetActive\$()
Description	Returns a String containing the name of the application.
Comments	If no application is active, the AppGetActive\$ function returns a zero-length string. You can use AppGetActive\$ to retrieve the name of the active application. You can then use this name in calls to routines that require an application name.
Example	<pre>Sub Main() n\$ = AppGetActive\$() AppMinimize n\$ End Sub</pre>

See Also	AppActivate (on page 313) (statement); WinFind (on page 785) (function).
Notes	This function returns a String containing the exact text appearing in the title bar of the active application's main window.

AppGetPosition (statement)

Syntax	AppGetPosition X,Y,width,height [,name\$]	
Description	Retrieves the position of the named application.	
Comments	The <code>AppGetPosition</code> statement takes the following parameters:	
	Parameter	Description
	X, Y	Names of Integer variables to receive the position of the application's window.
	width, height	Names of Integer variables to receive the size of the application's window.
	Name\$	String containing the name of the application. If the name\$ parameter is omitted, then the active application is used.
	<p>The x, y, width, and height variables are filled with the position and size of the application's window. If an argument is not a variable, then the argument is ignored, as in the following example, which only retrieves the x and y parameters and ignores the width and height parameters:</p> <pre>Dim x As Integer,y As Integer AppGetPosition x,y,0,0,"Program Manager"</pre>	
Example	<pre>Sub Main() Dim x As Integer,y As Integer Dim cx As Integer,cy As Integer AppGetPosition x,y,cx,cy,"Program Manager" End Sub</pre>	

	<pre>MsgBox "Program Manager is now minimized. Select OK to restore it." AppActivate "Program Manager" AppSetState state 'Restore it. End Sub</pre>
See Also	AppMaximize (on page 320) (statement); AppMinimize (on page 321) (statement); AppRestore (on page 323) (statement).
Notes	The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used.

AppHide (statement)

Syntax	AppHide [name\$]
Description	Hides the named application.
Comments	If the named application is already hidden, the AppHide statement will have no effect. The name\$ parameter is a String containing the name of the desired application. If it is omitted, then the AppHide statement hides the active application. AppHide generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.
Example	<p>This example hides Program Manager.</p> <pre>Sub Main() 'See whether Program Manager is running. If AppFind\$("Program Manager") = "" Then Exit Sub AppHide "Program Manager" MsgBox "Program Manager is now hidden. Press OK to show it once again." AppShow "Program Manager" End Sub</pre>
See Also	AppShow (on page 325) (statement).

Notes	The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used.
-------	---

AppList (statement)

Syntax	AppList AppNames\$()
Description	Fills an array with the names of all open applications.
Comments	The AppNames\$ parameter must specify either a zero- or one-dimensional dynamic String array or a one-dimensional fixed String array. If the array is dynamic, then it will be redimensioned to match the number of open applications. For fixed arrays, AppList first erases each array element, then begins assigning application names to the elements in the array. If there are fewer elements than will fit in the array, then the remaining elements are unused. The script returns a runtime error if the array is too small to hold the new elements. After calling this function, you can use LBound and UBound to determine the new size of the array.
Example	<p>This example minimizes all applications on the desktop.</p> <pre> Sub Main() Dim apps\$() AppList apps 'Check to see whether any applications were found. If ArrayDims(apps) = 0 Then Exit Sub For i = LBound(apps) To UBound(apps) AppMinimize apps(i) Next i End Sub </pre>
Notes	The name of an application is considered to be the exact text that appears in the title bar of the application's main window.

AppMaximize (statement)

Syn- tax	AppMaximize [name\$]
De- scrip- tion	Maximizes the named application.
Com- ments	The name\$ parameter is a String containing the name of the desired application. If it is omitted, then the AppMaximize function maximizes the active application.
Exam- ple	<pre> Sub Main() AppMaximize "Program Manager" 'Maximize Program Manager. If AppFind\$("NotePad") <> "" Then AppActivate "NotePad" 'Set the focus to NotePad. AppMaximize 'Maximize it. End If End Sub </pre>
See Also	AppMinimize (on page 321) (statement); AppRestore (on page 323) (statement); AppMove (on page 322) (statement); AppSize (on page 326) (statement); AppClose (on page 314) (statement).
Notes	If the named application is maximized or hidden, the AppMaximize statement will have no effect. The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. AppMaximize generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

AppMinimize (statement)

Syn- tax	AppMinimize [name\$]
De- scrip- tion	Minimizes the named application.

Comments	The name\$ parameter is a String containing the name of the desired application. If it is omitted, then the AppMinimize function minimizes the active application.
Example	<pre> Sub Main() AppMinimize "Program Manager" 'Maximize Program Manager. If AppFind\$("NotePad") <> "" Then AppActivate "NotePad" 'Set the focus to NotePad. AppMinimize 'Maximize it. End If End Sub </pre>
See Also	AppMaximize (on page 320) (statement); AppRestore (on page 323) (statement); AppMove (on page 322) (statement); AppSize (on page 326) (statement); AppClose (on page 314) (statement).
Notes	If the named application is minimized or hidden, the AppMinimize statement will have no effect. The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. AppMinimize generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

AppMove (statement)

Syntax	AppMove X, Y [,name\$]				
Description	Sets the upper left corner of the named application to a given location.				
Comments	The AppMove statement takes the following parameters:				
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>X, Y</td> <td>Integer coordinates specifying the upper left corner of the new location of the application, static to the upper left corner of the display.</td> </tr> </tbody> </table>	Parameter	Description	X, Y	Integer coordinates specifying the upper left corner of the new location of the application, static to the upper left corner of the display.
Parameter	Description				
X, Y	Integer coordinates specifying the upper left corner of the new location of the application, static to the upper left corner of the display.				

	<p>name String containing the name of the application to move. If this parameter is omitted, then the active application is moved.</p> <p>\$</p>
Example	<p>This example activates Program Manager, then moves it 10 pixels to the right.</p> <pre> Sub Main() Dim x%,y% AppActivate "Program Manager" 'Activate Program Manager. AppGetPosition x%,y%,0,0 'Retrieve its position. x% = x% + Screen.TwipsPerPixelX * 10 'Add 10 pixels. AppMove x% + 10,y% 'Nudge it 10 pixels to the right. End Sub </pre>
See Also	<p>AppMaximize (on page 320) (statement); AppMinimize (on page 321) (statement); AppRestore (on page 323) (statement); AppSize (on page 326) (statement); AppClose (on page 314) (statement).</p>
Note	<p>If the named application is maximized or hidden, the AppMove statement will have no effect. The X and Y parameters are specified in twips. AppMove will accept X and Y parameters that are off the screen. The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. AppMove generates a runtime error if the named application is not enabled, as is the case if that application is currently displaying a modal dialog box.</p>

AppRestore (statement)

Syntax	AppRestore [name\$]
Description	Restores the named application.
Comments	The name\$ parameter is a String containing the name of the application to restore. If this parameter is omitted, then the active application is restored.
Example	<p>This example minimizes Program Manager, then restores it.</p> <pre> Sub Main() If AppFind\$("Program Manager") = "" Then Exit Sub </pre>

	<pre> AppActivate "Program Manager" AppMinimize "Program Manager" MsgBox "Program Manager is now minimized. Press OK to restore it." AppRestore "Program Manager" End Sub </pre>
See Also	AppMaximize (on page 320) (statement); AppMinimize (on page 321) (statement); AppMove (on page 322) (statement); AppSize (on page 326) (statement); AppClose (on page 314) (statement).
Notes	<p>The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. AppRestore will have an effect only if the main window of the named application is either maximized or minimized. AppRestore will have no effect if the named window is hidden. AppRestore generates a runtime error if the named application is not enabled, as is the case if that application is currently displaying a modal dialog box.</p>

AppSetState (statement)

Syntax	AppSetState newstate [,name\$]		
Description	Maximizes, minimizes, or restores the named application, depending on the value of newstate.		
Comments	The AppSetState statement takes the following parameters:		
	Parameter	Description	
	Newstate	Integer specifying the new state of the window. It can be any of the following values.	
		Value	Description
		ebMaximized	The named application is maximized.
		ebMinimized	The named application is minimized.
		ebRestored	The named application is restored.

	<p>Name\$ String containing the name of the application to change. If this parameter is omitted, then the active application is used.</p>
Example	<p>This example saves the state of Program Manager, changes it, then restores it to its original setting.</p> <pre> Sub Main() If AppFind\$("Program Manager") = "" Then MsgBox "Can't find Program Manager." Exit Sub End If AppActivate "Program Manager" 'Activate Program Manager. state = AppGetState 'Save its state. AppMinimize 'Minimize it. MsgBox "Program Manager is now minimized. Select OK to restore it." AppActivate "Program Manager" AppSetState state 'Restore it. End Sub </pre>
See Also	<p>AppGetState (on page 318) (function); AppRestore (on page 323) (statement); AppMaximize (on page 320) (statement); AppMinimize (on page 321) (statement)</p>
Notes	<p>The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used.</p>

AppShow (statement)

Syntax	<p>AppShow [name\$]</p>
Description	<p>Makes the named application visible.</p>
Comments	<p>The name\$ parameter is a String containing the name of the application to show. If this parameter is omitted, then the active application is shown.</p>
Example	<p>This example hides Program Manager.</p>

	<pre> Sub Main() 'See whether Program Manager is running. If AppFind\$("Program Manager") = "" Then Exit Sub AppHide "Program Manager" MsgBox "Program Manager is now hidden. Press OK to show it once again." AppShow "Program Manager" End Sub </pre>
See Also	AppHide (on page 319) (statement).
Notes:	If the named application is already visible, AppShow will have no effect. The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. AppShow generates a runtime error if the named application is not enabled, as is the case if that application is displaying a modal dialog box.

AppSize (statement)

Syntax	AppSize width,height [,name\$]	
Description	Sets the width and height of the named application.	
Comments	The AppSize statement takes the following parameters:	
	Parameter	Description
	Width, height	Integer coordinates specifying the new size of the application.
	Name\$	String containing the name of the application to resize. If this parameter is omitted, then the active application is used.
Example	This example enlarges the active application by 10 pixels in both the vertical and horizontal directions.	

	<pre> Sub Main() Dim w%,h% AppGetPosition 0,0,w%,h% 'Get current width/height. x% = x% + Screen.TwipsPerPixelX * 10 'Add 10 pixels. y% = y% + Screen.TwipsPerPixelY * 10 'Add 10 pixels. AppSize w%,h% 'Change to new size. End Sub </pre>
See Also	AppMaximize (on page 320) (statement); AppMinimize (on page 321) (statement); AppRestore (on page 323) (statement); AppMove (on page 322) (statement); AppClose (on page 314) (statement).
Note	The width and height parameters are specified in twips. This statement will only work if the named application is restored (i.e., not minimized or maximized). The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used. A runtime error results if the application being resized is not enabled, which is the case if that application is displaying a modal dialog box when an AppSize statement is executed.

AppType (function)

Syntax	AppType [(name\$)]				
Description	Returns an Integer indicating the executable file type of the named application:				
	<table border="1"> <tr> <td>ebDos</td> <td>DOS executable</td> </tr> <tr> <td>ebWindows</td> <td>Windows executable</td> </tr> </table>	ebDos	DOS executable	ebWindows	Windows executable
ebDos	DOS executable				
ebWindows	Windows executable				
Comments	The name\$ parameter is a String containing the name of the application. If this parameter is omitted, then the active application is used.				
Example	This example creates an array of strings containing the names of all the running Windows applications. It uses the AppType command to determine whether an application is a Windows application or a DOS application.				


```

Sub Main()

    Dim apps$,wapps$()

    AppList apps      'Retrieve a list of all Windows and DOS apps.

    If ArrayDims(apps) = 0 Then

        MsgBox "There are no running applications."

        Exit Sub

    End If

    'Create an array to hold only the Windows apps.

    ReDim wapps$(UBound(apps))

    n = 0 'Copy the Windows apps from one array to the target array.

    For i = LBound(apps) to UBound(apps)

        If AppType(apps(i)) = ebWindows Then

            wapps(n) = apps(i)

            n = n + 1

        End If

    Next I

    If n = 0 Then      'Make sure at least one Windows app was found.

        MsgBox "There are no running Windows applications."

        Exit Sub

    End If

    ReDim Preserve wapps(n - 1) 'Resize to hold the exact number.

    'Let the user pick one.

    index% = SelectBox("Windows Applications","Select a Windows application:",wapps)

End Sub

```

Notes	The name\$ parameter is the exact string appearing in the title bar of the named application's main window. If no application is found whose title exactly matches name\$, then a second search is performed for applications whose title string begins with name\$. If more than one application is found that matches name\$, then the first application encountered is used.
-------	---

ArrayDims (function)

Syn- tax	ArrayDims (arrayvariable)
De- scrip- tion	Returns an Integer containing the number of dimensions of a given array.

Com- ments	This function can be used to determine whether a given array contains any elements or if the array is initially created with no dimensions and then redimensioned by another function, such as the FileList function, as shown in the following example.
Exam- ple	<p>This example allocates an empty (null-dimensional) array; fills the array with a list of filenames, which resizes the array; then tests the array dimension and displays an appropriate message.</p> <pre data-bbox="305 464 1427 877"> Sub Main() Dim f\$() FileList f\$, "c:*.bat" If ArrayDims(f\$) = 0 Then MsgBox "The array is empty." Else MsgBox "The array size is: " & (UBound(f\$) - LBound(f\$) + 1) End If End Sub </pre>
See Also	LBound (on page 581) (function); UBound (on page 765) (function); Arrays (topic)

Arrays (topic)

Declaring Array Variables Arrays in a Basic Control Engine script are declared using any of the following statements:

```
Dim
Public
Private
```

For example:

```
Dim a(10) As Integer
Public LastNames(1 to 5,-2 to 7) As Variant
Private
```

Arrays of any data type can be created, including **Integer**, **Long**, **Single**, **Double**, **Boolean**, **Date**, **Variant**, **Object**, user-defined structures, and data objects. The lower and upper bounds of each array dimension must be within the following range:

```
-32768 <= bound <= 32767
```

Arrays can have up to 60 dimensions. Arrays can be declared as either fixed or dynamic, as described below.

Fixed Arrays The dimensions of fixed arrays cannot be adjusted at execution time. Once declared, a fixed array will always require the same amount of storage. Fixed arrays can be declared with the **Dim**, **Private**, or **Public** statement by supplying explicit dimensions. The following example declares a fixed array of ten strings:

```
Dim a(10) As String
```

Fixed arrays can be used as members of user-defined data types. The following example shows a structure containing fixed-length arrays:

```
Type Foo
    rect(4) As Integer
    colors(10) As Integer
End Type
```

Only fixed arrays can appear within structures.

Dynamic Arrays Dynamic arrays are declared without explicit dimensions, as shown below:

```
Public Ages() As Integer
```

Dynamic arrays can be resized at execution time using the **Redim** statement:

```
Redim Ages$(100)
```

Subsequent to their initial declaration, dynamic arrays can be redimensioned any number of times. When redimensioning an array, the old array is first erased unless you use the **Preserve** keyword, as shown below:

```
Redim Preserve Ages$(100)
```

Dynamic arrays cannot be members of user-defined data types.

Passing Arrays Arrays are always passed by reference.

Querying Arrays The following table describes the functions used to retrieve information about arrays.

Use this function	to
LBound	Retrieve the lower bound of an array. A runtime error is generated if the array has no dimensions.
UBound	Retrieve the upper bound of an array. A runtime error is generated if the array has no dimensions.
ArrayDims	Retrieve the number of dimensions of an array. This function returns 0 if the array has no dimensions

Operations on Arrays

The following table describes the function that operate on arrays:

Use this command	to
ArraySort	Sort an array of integers, longs, singles, doubles, currency, Booleans, dates, or variants.
FileList	Fill an array with a list of files in a given directory.
DiskDrives	Fill an array with a list of valid drive letters.
AppList	Fill an array with a list of running applications.
SelectBox	Display the contents of an array in a list box.
PopupMenu	Display the contents of an array in a pop-up menu.
ReadIniSection	Fill an array with the item names from a section in an ini file.
FileDirs	Fill an array with a list of subdirectories.
Erase	Erase all the elements of an array.
ReDim	Establish the bounds and dimensions of an array.
Dim	Declare an array.

ArraySort (statement)

Syntax	ArraySort array()
Description	Sorts a single-dimensional array in ascending order.
Comments	If a string array is specified, then the routine sorts alphabetically in ascending order using case-sensitive string comparisons. If a numeric array is specified, the ArraySort statement sorts smaller numbers to the lowest array index locations. The script generates a runtime error if you specify an array with more than one dimension. When sorting an array of variants, the following rules apply:

	<ul style="list-style-type: none"> • A runtime error is generated if any element of the array is an object. • String is greater than any numeric type. • Null is less than String and all numeric types. • Empty is treated as a number with the value 0. <p>String comparison is case-sensitive (this function is not affected by the Option Compare setting).</p>
Example	<p>This example dimensions an array and fills it with filenames using FileList, then sorts the array and displays it in a select box.</p> <pre> Sub Main() Dim f\$() FileList f\$,"c:*.*" ArraySort f\$ r% = SelectBox("Files","Choose one:",f\$) End Sub </pre>
See Also	<p>ArrayDims (on page 328) (function); LBound (on page 581) (function); UBound (on page 765) (function)</p>

Asc, AscB, AscW (functions)

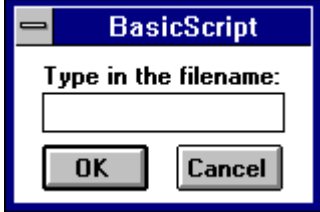
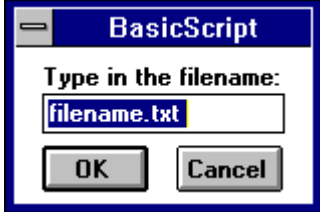
Syntax	<code>Asc (string) AscB (string) AscW (string)</code>
Description	<p>Returns an Integer containing the numeric code for the first character of string.</p>
Comments	<p>This function returns the character value of the first character of string. On single-byte systems, this function returns a number between 0 and 255, whereas on MBCS systems, this function returns a number between -32768 and 32767. On wide platforms, this function returns the <code>MBCS</code> character code after converting the wide character to <code>MBCS</code>. To return the value of the first byte of a string, use the <code>AscB</code> function. This function is used when you need the value of the first byte of a string known to contain byte data rather than character data. On single-byte systems, the <code>AscB</code> function is identical to the <code>Asc</code> function. On platforms where BasicScript uses wide string internally (such as Win32), the <code>AscW</code> function returns the character value native to that platform. For example, on Win32 platforms, this function returns the UNICODE character code. On sin-</p>

	<p>gle-byte and MBCS platforms, the <code>AscW</code> function is equivalent to the <code>Asc</code> function. The following table summarizes the values returned by these functions:</p>		
	Function	String Format	Returns value of the:
	<code>Asc</code>		First byte of string (between 0 and 255)
		MBCS	First character of string (between -32769 and 32767)
		Wide	First character of string after conversion to MBCS.
	<code>AscB</code>		First byte of string .
		MBCS	First byte of string .
		Wide	First byte of string .
	<code>AscW</code>		Same as <code>Asc</code> .
		MBCS	Same as <code>Asc</code> .
		Wide	Wide character native to the operating system.
Example	<p>This example fills an array with the ASCII values of the string <code>s</code> components and displays the result.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() s\$ = InputBox("Please enter a string.", "Enter String") If s\$ = "" Then End 'Exit if no string entered. msg1 = "" For i = 1 To Len(s\$) msg1 = msg1 & Asc(Mid(s\$,i,1)) & crlf Next i MsgBox "The Asc values of the string are:" & msg1 End Sub </pre>		
See Also	<p>Chr (on page 362), Chr\$ (on page 362) (functions).</p>		

AskBox, AskBox\$ (functions)


Syntax	<code>AskBox[\$](prompt\$ [, [default\$] [, [title\$][, helpfile, context]])</code>
--------	---

De- scrip- tion	Displays a dialog box requesting input from the user and returns that input as a String .	
Com- ments	The <code>AskBox/AskBox\$</code> functions take the following parameters:	
	Parameter	Description
	prompt\$	String containing the text to be displayed above the text box. The dialog box is sized to the appropriate width depending on the width of prompt\$. A runtime error is generated if prompt\$ is Null.
	default\$	String containing the initial content of the text box. The user can return the default by immediately selecting OK. A runtime error is generated if default\$ is Null.
	title\$	String specifying the title of the dialog. If missing, then the default title is used.
	helpfile	Name of the file containing context-sensitive help for this dialog. If this parameter is specified, then context must also be specified.
	context	Number specifying the ID of the topic within helpfile for this dialog's help. If this parameter is specified, then helpfile must also be specified.
	Function	Returns
	<code>AskBox\$</code>	String containing the input typed by the user in the text box. A zero-length string is returned if the user selects Cancel.
	<code>AskBox</code>	String variant containing the input typed by the user in the text box. An Empty variant is returned if the user selects Cancel.
	When the dialog box is displayed, the text box has the focus. The user can type a maximum of 255 characters into the text box displayed by AskBox\$. If both the helpfile and context parameters are specified, then a Help button is added in addition to the OK and Cancel buttons. Context-sensitive help can be invoked by selecting this button or using the help key (F1). Invoking help does not remove the dialog.	
	s\$ = AskBox\$ (" Type in the filename:")	

	 <p>s\$ = AskBox\$ ("Type in the filename:","filename.txt")</p> 
<p>Example</p>	<p>This example asks the user to enter a filename and then displays what he or she has typed.</p> <pre>Sub Main() s\$ = AskBox\$("Type in the filename:") MsgBox "The filename was: " & s\$ End Sub</pre>
<p>See Also</p>	<p>MsgBox (on page 614) (statement); AskPassword\$ (function); InputBox, InputBox\$ (on page 562) (functions); OpenFilename\$ (on page 646) (function); SaveFilename\$ (on page 699) (function); SelectBox (on page 709) (function).</p>
<p>Note</p>	<p>The text in the dialog box is displayed in 8-point MS Sans Serif.</p>

AskPassword, AskPassword\$ (functions)

<p>Syntax</p>	<pre>AskPassword[\$](prompt\$ [,title\$] [,helpfile,context])</pre>		
<p>Description</p>	<p>Returns a String containing the text that the user typed.</p>		
<p>Comments</p>	<p>Unlike the <code>AskBox/AskBox\$</code> functions, the user sees asterisks in place of the characters that are actually typed. This allows the hidden input of passwords. The <code>AskPassword/AskPassword\$</code> functions take the following parameters:</p>		
	<table border="1"> <thead> <tr> <th data-bbox="289 1791 553 1837">Parameter</th> <th data-bbox="553 1791 1422 1837">Description</th> </tr> </thead> </table>	Parameter	Description
Parameter	Description		

	prompt\$	String containing the text to be displayed above the text box. The dialog box is sized to the appropriate width depending on the width of prompt\$. A runtime error is generated if prompt\$ is Null.
	title\$	String specifying the title of the dialog. If missing, then the default title is used.
	helpfile	Name of the file containing context-sensitive help for this dialog. If this parameter is specified, then context must also be specified.
	context	Number specifying the ID of the topic within helpfile for this dialog's help. If this parameter is specified, then helpfile must also be specified.
When the dialog box is first displayed, the text box has the focus. A maximum of 255 characters can be typed into the text box.		
	Function	Returns
	AskPassword\$	text typed into the text box, up to a maximum of 255 characters. A zero-length string is returned if the user selects Cancel.
	AskPassword	String variant. An <code>Empty</code> variant is returned if the user selects Cancel.
If both the helpfile and context parameters are specified, then a Help button is added in addition to the OK and Cancel buttons. Context-sensitive help can be invoked by selecting this button or using the help key (F1 on most platforms). Invoking help does not remove the dialog.		
<p>s\$ = AskPassword\$ ("Type in the password:")</p> 		
Example	<pre>Sub Main() s\$ = AskPassword\$("Type in the password:") MsgBox "The password entered is: " & s\$ End Sub</pre>	
See Also	MsgBox (on page 614) (statement); AskBox\$ (on page 333) (function); InputBox , InputBox\$ (on page 562) (functions); OpenFilename\$ (on page 646) (function); SaveFilename\$ (on page 699) (function); SelectBox (on page 709) (function); AnswerBox (on page 310) (function).	

Notes	The text in the dialog box is displayed in 8-point MS Sans Serif.
-------	---

Atn (function)

Syntax	Atn (number)
Description	Returns the angle (in radians) whose tangent is number.
Comments	Some helpful conversions: <ul style="list-style-type: none"> • Pi (3.1415926536) radians = 180 degrees. • radian = 57.2957795131 degrees. • degree = .0174532925 radians.
Example	<p>This example finds the angle whose tangent is 1 (45 degrees) and displays the result.</p> <pre>Sub Main() a# = Atn(1.00) MsgBox "1.00 is the tangent of " & a# & " radians (45 degrees)."</pre> <p>End Sub</p>
See Also	Tan (on page 751) (function); Sin (on page 718) (function); Cos (on page 385) (function).

B

B

Basic.Architecture\$ (property)
Basic.Capability (method)
Basic.CodePage (property)
Basic.Eoln\$ (property)
Basic.FreeMemory (property)
Basic.HomeDir\$ (property)
Basic.Locale\$ (property)

Basic.OperatingSystem\$ (Property)
Basic.OperatingSystemVendor\$
Basic.OperatingSystemVersion\$
Basic.OS (property)
Basic.Pathseparator\$ (property)
Basic.Processor\$ (Property)
Basic.ProcessorCount\$ (property)
Basic.Version\$ (property)
Beep (statement)
Begin Dialog (statement)
Boolean (data type)
ByRef (keyword)
ByVal (keyword)

Basic.Architecture\$ (property)

Syntax	<code>Basic.Architecture\$</code>	
Description	Returns a String containing the CPU architecture on which BasicScript is executing.	
Comments	The following table describes what <code>Basic.Architecture\$</code> returns on:	
	Win32	Intel, MIPS, Alpha AXP, or PowerPC
	The <code>Basic.Architecture\$</code> property returns an empty string if the architecture cannot be determined by BasicScript.	
Example	<pre> ' 'Print the CPU architecture... ' Sub Main() </pre>	

	<pre>MsgBox Basic.Architecture\$ End Sub</pre>
See Also	Basic.Processor\$ (on page 346) (property), Basic.ProcessorCount (on page 347) (property)

Basic.Capability (method)

Syntax	<code>Basic.Capability(which)</code>	
Description	Returns True if the specified capability exists on the current platform; returns False otherwise.	
Comments	The which parameter is an Integer specifying the capability for which to test. It can be any of the following values:	
	Value	Returns True If the Platform Supports
	1	Disk drives
	2	System file attribute (ebSystem)
	3	Hidden file attribute (ebHidden)
	4	Volume label file attribute (ebVolume)
	5	Archive file attribute (ebArchive)
	6	Denormalized floating-point math
	7	File locking (i.e., the Lock and Unlock statements)
	8	Big endian byte ordering
Example	<p>This example tests to see whether your current platform supports disk drives and hidden file attributes and displays the result.</p> <pre>Sub Main() msg1 = "This operating system " If Basic.Capability(1) Then msg1 = msg1 & "supports disk drives." Else msg1 = msg1 & "does not support disk drives." End If End Sub</pre>	

	<pre>MsgBox msg1 End Sub</pre>
See Also	Basic.OS (on page 345) (property)

Basic.CodePage (property)

Syntax	<code>Basic.CodePage</code>
Description	Returns an Integer representing the code page for the current locale.
Comments	<code>Basic.CodePage</code> returns ANSI code page for the current locale, such as 437 for MS-DOS Latin US or 932 for Japanese.
Example	<pre>Sub Main If Basic.OS = ebWin16 And Basic.CodePage = 437 Then MsgBox "Running US Windows" Else if Basic.OS = ebWin32 And Basic.CodePage = 932 Then MsgBox "Japanese XP" End If End Sub</pre>
See Also	Basic.Locale\$ (on page 342) (property); Basic.OS (on page 345) (property)

Basic.Eoln\$ (property)

Syntax	Basic.Eoln\$
Description	Returns a String containing the end-of-line character sequence appropriate to the current platform.
Comments	This string will be either a carriage return, a carriage return/line feed, or a line feed.
Example	This example writes two lines of text in a message box.

	<pre>Sub Main() MsgBox "This is the first line of text." & Basic.Eoln\$ & "This is the second line of text." End Sub</pre>
See Also	Basic.PathSeparator\$ (on page 346) (property).

Basic.FreeMemory (property)

Syntax	Basic.FreeMemory
Description	Returns a Long representing the number of bytes of free memory in the script's data space.
Comments	This function returns the size of the largest free block in the script's data space. Before this number is returned, the data space is compacted, consolidating free space into a single contiguous free block. The script's data space contains strings and dynamic arrays.
Example	<p>This example displays free memory in a dialog box.</p> <pre>Sub Main() MsgBox "The largest free memory block is: " & Basic.FreeMemory End Sub</pre>
See Also	System.TotalMemory (on page 748) (property); System.FreeMemory (on page 747) (property); System.FreeResources (on page 747) (property); Basic.FreeMemory (on page 341) (property).

Basic.HomeDir\$ (property)

Syntax	<code>Basic.HomeDir\$</code>
Description	Returns the path to the basic script runtime engine components, e.g. c:\Program Files\Proficy\Proficy CIMPLICITY\exe.
Comments	This method is used to find the HMI/SCADA CIMPLICITY exe directory.
Example	<p>This example assigns the home directory to HD and displays it.</p> <pre>Sub Main() hd\$ = Basic.HomeDir\$</pre>

	<pre>MsgBox "The Basic Control Engine home directory is: " & hd\$ End Sub</pre>
See Also	System.WindowsDirectory\$ (on page 749) (property).

Basic.Locale\$ (property)

Syntax	<code>Basic.Locale\$</code>								
Description	Returns a String containing the locale under which BasicScript is running.								
Comments	The locale helps you identify information about your environment, such as the date formats, time format, and other country-sensitive information. The following table describes the returned value from Basic.Locale\$ on the Win32 platform.								
	Returns a string in the format:								
	<code>abbrevlang, langid, nativelang, englang</code>								
	<table border="1"> <tr> <td><code>abbrevlang</code></td> <td>Three-letter name of the language. This name is formed by taking the two-letter language abbreviation as found in the ISO Standard 639 and adding a third letter, as appropriate, to indicate the sublanguage.</td> </tr> <tr> <td><code>langid:</code></td> <td>Language ID as defined by the operating system.</td> </tr> <tr> <td><code>nativelang</code></td> <td>Native name of the language.</td> </tr> <tr> <td><code>englang:</code></td> <td>Full English name of the language as defined by ISO standard 639.</td> </tr> </table>	<code>abbrevlang</code>	Three-letter name of the language. This name is formed by taking the two-letter language abbreviation as found in the ISO Standard 639 and adding a third letter, as appropriate, to indicate the sublanguage.	<code>langid:</code>	Language ID as defined by the operating system.	<code>nativelang</code>	Native name of the language.	<code>englang:</code>	Full English name of the language as defined by ISO standard 639.
<code>abbrevlang</code>	Three-letter name of the language. This name is formed by taking the two-letter language abbreviation as found in the ISO Standard 639 and adding a third letter, as appropriate, to indicate the sublanguage.								
<code>langid:</code>	Language ID as defined by the operating system.								
<code>nativelang</code>	Native name of the language.								
<code>englang:</code>	Full English name of the language as defined by ISO standard 639.								
Example	<pre>'This example checks to see if we are running in a Japanese 'version of Windows. Sub Main If Basic.OS = ebWin16 And Item\$(Basic.Locale\$,1) = "jpn" Then MsgBox "Running Windows on a Japanese computer." End If End Sub</pre>								

See Also	Basic.OS (on page 345) (property) , Basic.CodePage (on page 340) (property)
----------	---

Basic.OperatingSystem\$ (property)

Syntax	<code>Basic.OperatingSystem\$</code>
Description	Returns a String containing the name of the operating system.
Comments	The value returned by this function for the Win32 operating systems is Win32s.
	The version of the operating system is determined by calling <code>Basic.OperatingSystemVersion\$</code> .
Example	<pre> 'This script checks the Windows version for special networking 'capabilities. ' Sub Main() If Basic.OS = ebWin16 Then If Basic.OperatingSystem\$ = "Windows" Then MsgBox "Special networking capabilities aren't present." ElseIf Basic.OperatingSystem\$ = "Windows for Workgroups" Then MsgBox "Network capabilities are present." End If End Sub </pre>
See Also	Basic.OperatingSystemVendor\$ (on page 343) (property), Basic.OperatingSystemVersion\$ (on page 344) (property), Basic.OS (on page 345) (property)

Basic.OperatingSystemVendor\$ (property)

Syntax	<code>Basic.OperatingSystemVendor\$</code>
Description	Returns a String containing the version of the operating system under which BasicScript is running.

Com-ments	For the Win32 platform, <code>Basic.OperatingSystemVendor\$</code> returns, Microsoft.
Example	<pre> ' 'The following example prints the operating system vendor ' Sub Main MsgBox "The manufacturer of the operating system is: " & _ Basic.OperatingSystemVendor\$ End Sub </pre>
See Also	Basic.OperatingSystem\$ (on page 343) (property), Basic.OperatingSystemVersion\$ (on page 344) (property), Basic.OS (on page 345) (property)

Basic.OperatingSystemVersion\$ (property)

Syntax	<code>Basic.OperatingSystemVersion\$</code>	
De-scrip-tion	Returns a String containing the version of the operating system under which BasicScript is running.	
Com-ments	The version number is returned in the following format: <code>major.minor.buildnumber</code> The parts of the version number are as follows.	
	Part	Identifies the:
	major	Major version number of the operating system.
	minor	Minor version number of the operating system.
	buildnumber	Build number of the operating system.
Exam-ple	<pre> ' 'This example checks the Windows version to ensure that a 'feature is supported. ' Sub Main If Basic.OperatingSystem\$ = "Windows" </pre>	

	<pre> If Basic.OperatingSystemVersion\$ <= 2000 Then MsgBox "That feature is not supported." Else MsgBox "Windows version 2000 or greater" End If End If End Sub </pre>
See Also	Basic.OperatingSystem\$ (property), Basic.OperatingSystemVendor\$ (property), Basic.OS (property)

Basic.OS (property)

Syntax	<code>Basic.OS</code>		
Description	Returns an Integer indicating the current platform.		
Comments	Value	Constant	Platform
	2	<code>ebWin32</code>	Windows XP Windows 2003
	The value returned is not necessarily the platform under which the Basic Control Language script is running but rather an indicator of the platform for which the script was created.		
Example	<p>This example determines the operating system for which this version was created and displays the appropriate message.</p> <pre> Sub Main() Select Case Basic.OS Case ebWin32 s = "Windows XP" Case Else s = "not Windows XP" End Select MsgBox "You are currently running " & s End Sub </pre>		

Basic.PathSeparator\$ (property)

Syntax	<code>Basic.PathSeparator\$</code>
Description	Returns a String containing the path separator appropriate for the current platform.
Comments	The returned string is any one of the following characters: / (slash), \ (back slash), : (colon)
Example	<pre>Sub Main() MsgBox "The path separator for this platform is: " & Basic.PathSeparator\$ End Sub</pre>
See Also	Basic.Eoln\$ (on page 340) (property)

Basic.Processor\$ (property)

Syntax	<code>Basic.Processor\$</code>	
Description	Returns a String containing the name of the CPU in the computer on which BasicScript is running.	
Comments	Sample values returned for Win32 platforms include:	
	Platform	Sample Value returned
	Intel	80386 80486 Pentium
	MIPS	The string "Rx" such as R4000
	Alpha	321064 321066 321164
	PowerPC	<pre>601 603 604 603+ 604+ 620</pre>
Example	<pre> ' 'This example prints the CPU of the computer on which 'BasicScript is executing.</pre>	

	<pre> ' Sub Main() MsgBox "Processor = " & Basic.Processor\$ End Sub </pre>
See Also	Basic.ProcessorCount (on page 347) (property)
Note	You can retrieve the number of processors within the computer using the <code>Basic.ProcessorCount</code> property.

Basic.ProcessorCount\$ (property)

Syntax	<code>Basic.ProcessorCount</code>
Description	Returns the number of CPUs installed on the computer on which BasicScript is running.
Comments	<code>Basic.ProcessorCount\$</code> returns 1 if the CPU has only one processor or is otherwise incapable of containing more than one processor.
Example	<pre> ' 'Print the number of processors in the computer. ' Sub Main() MsgBox "There are " & Basic.ProcessorCount & _ " processor(s) in the computer." End Sub </pre>
See Also	Basic.Processor\$ (on page 346) (property)
Note	The <code>Basic.Processor\$</code> property determines the type of processor.

Basic.Version\$ (Property)

Syntax	<code>Basic.Version\$</code>
--------	------------------------------

Description	Returns a String containing the version of Basic Control Engine.
Comments	This function returns the major and minor version numbers in the format major.minor.Build-Number, as in "2.00.30."
Example	This example displays the current version of the Basic Control Engine. <pre>Sub Main() MsgBox "Version " & Basic.Version\$ & " of Basic Control Engine is running" End Sub</pre>

Beep (statement)

Syntax	<code>Beep</code>
Description	Makes a single system beep.
Example	This example causes the system to beep five times and displays a reminder message. <pre>Sub Main() For i = 1 To 5 Beep Sleep 200 Next i MsgBox "You have an upcoming appointment!" End Sub</pre>

Begin Dialog (statement)

Syntax	<code>Begin Dialog</code> DialogName [x],[y],width,height,title\$ [[.DlgProc] [[PicName\$] [,style]]] Dialog Statements End Dialog
Description	Defines a dialog box template for use with the Dialog statement and function.
Comments	A dialog box template is constructed by placing any of the following statements between the <code>Begin Dialog</code> and <code>End Dialog</code> statements (no other statements besides comments can appear within a dialog box template):

	Picture	OptionButton	OptionGroup
	CancelButton	Text	TextBox
	GroupBox	DropListBox	ListBox
	ComboBox	CheckBox	PictureButton
	PushButton	OKButton	
The <code>Begin Dialog</code> statement requires the following parameters:			
	Parameter	Description	
	x, y	Integer coordinates specifying the position of the upper left corner of the dialog box static to the parent window. These coordinates are in dialog units. If either coordinate is unspecified, then the dialog box will be centered in that direction on the parent window.	
	width, height	Integer coordinates specifying the width and height of the dialog box (in dialog units).	
	Dialog-Name	Name of the dialog box template. Once a dialog box template has been created, a variable can be dimensioned using this name.	
	title\$	String containing the name to appear in the title bar of the dialog box. If this parameter specifies a zero-length string, then the name "Basic Control Engine" is used.	
	.DlgProc	Name of the dialog function. The routine specified by <code>.DlgProc</code> will be called by the script when certain actions occur during processing of the dialog box. (See <code>DlgProc [prototype]</code> for additional information about dialog functions.) If this omitted, then the script processes the dialog box using the default dialog box processing behavior.	
	style	Specifies extra styles for the dialog. It can be any of the following values:	
		Value	Meaning
		0	Dialog does not contain a title or close box.
		1	Dialog contains a title and no close box.
		2 (or omitted)	Dialog contains both the title and close box.

The script generates an error if the dialog box template contains no controls. A dialog box template must have at least one **PushButton**, **OKButton**, or **CancelButton** statement. Otherwise, there will be no way to close the dialog box. Dialog units are defined as $\frac{1}{4}$ the width of the font in the horizontal direction and $\frac{1}{8}$ the height of the font in the vertical direction. Any number of user dialog boxes can be created, but each one must be created using a different name as the DialogName. Only one user dialog box may be invoked at any time. Expression Evaluation within the Dialog Box Template The **Begin Dialog** statement creates the template for the dialog box. Any expression or variable name that appears within any of the statements in the dialog box template is not evaluated until a variable is dimensioned of type DialogName. The following example shows this behavior:

```
Sub Main()

  MyTitle$ = "Hello, World"

  Begin Dialog MyTemplate 16,32,116,64,MyTitle$

    OKButton 12,40,40,14

  End Dialog

  MyTitle$ = "Sample Dialog"

  Dim dummy As MyTemplate

  rc% = Dialog(dummy)

End Sub
```

The above example creates a dialog box with the title " `Sample Dialog` ". Expressions within dialog box templates cannot reference external subroutines or functions. All controls within a dialog box use the same font. The fonts used for text and text box control can be changed explicitly by setting the font parameters in the **Text** and `TextBox` statements. A maximum of 128 fonts can be used within a single dialog, although the practical limitation may be less.

Example This example creates an exit dialog box.

```
Sub Main()

  Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit"

    Text 4,8,108,8,"Are you sure you want to exit?"

    CheckBox 32,24,63,8,"Save Changes",.SaveChanges

    OKButton 12,40,40,14

    CancelButton 60,40,40,14

  End Dialog

  Dim QuitDialog As QuitDialogTemplate

  rc% = Dialog(QuitDialog)

  Select Case rc%

  Case -1
```

	<pre> MsgBox "OK was pressed!" Case 1 MsgBox "Cancel was pressed!" End Select End Sub </pre>
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); End Dialog (on page 487) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); DlgProc (on page 440) (function).
Note	Within user dialog boxes, the default font is 8-point MS Sans Serif.

Boolean (data type)

Syntax	<code>Boolean</code>
Description	A data type capable of representing the logical values TRUE and FALSE .
Comments	<p>Boolean variables are used to hold a binary value—either TRUE or FALSE. Variables can be declared as Boolean using the Dim , Public , or Private statement. Variants can hold Boolean values when assigned the results of comparisons or the constants TRUE or FALSE. Internally, a Boolean variable is a 2-byte value holding -1 (for TRUE) or 0 (for FALSE). Any type of data can be assigned to Boolean variables. When assigning, non-0 values are converted to TRUE , and 0 values are converted to FALSE. When appearing as a structure member, Boolean members require 2 bytes of storage. When used within binary or random files, 2 bytes of storage are required. When passed to external routines, Boolean values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack. There is no type-declaration character for Boolean variables.</p>
	Boolean variables that have not yet been assigned are given an initial value of False .
See Also	Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Integer (on page 566) (data type); Long (on page 598) (data type); Object (on page 633) (data type); Single (on page 718) (data type); String (on page 742) (data type);

[Variant \(on page 771\)](#) (data type); [DefType \(on page 421\)](#) (statement); [CBool \(on page 356\)](#) (function); [True \(on page 760\)](#) (constant); [False \(on page 511\)](#) (constant).

ByRef (keyword)

Syntax	<code>..., ByRef parameter,...</code>
Description	Used within the <code>Sub...End Sub</code> , <code>Function...End Function</code> , or <code>Declare</code> statement to specify that a given parameter can be modified by the called routine.
Comments	<p>Passing a parameter by reference means that the caller can modify that variable's value. Unlike the <code>ByVal</code> keyword, the <code>ByRef</code> keyword cannot be used when passing a parameter. The absence of the <code>ByVal</code> keyword is sufficient to force a parameter to be passed by reference:</p> <pre> MySub ByVal i '-- Pass i by value. MySub ByRef i '-- Illegal (will not compile). MySub i '-- Pass i by reference. </pre>
Example	<pre> Sub Test(ByRef a As Variant) a = 14 End Sub Sub Main() b = 12 Test b MsgBox "The ByRef value is: " & b '-- Displays 14. End Sub </pre>
See Also	() (on page 297) (keyword), ByVal (on page 352) (keyword).

ByVal (keyword)

Syntax	<code>...ByVal parameter...</code>
Description	Forces a parameter to be passed by value rather than by reference.

Com- ments	<p>The <code>ByVal</code> keyword can appear before any parameter passed to any function, statement, or method to force that parameter to be passed by value. Passing a parameter by value means that the caller cannot modify that variable's value. Enclosing a variable within parentheses has the same effect as the <code>ByVal</code> keyword:</p> <pre> Foo ByVal i 'Forces i to be passed by value. Foo(i) 'Forces i to be passed by value. </pre>
	<p>When calling external statements and functions (that is, routines defined using the <code>Declare</code> statement), the <code>ByVal</code> keyword forces the parameter to be passed by value regardless of the declaration of that parameter in the <code>Declare</code> statement. The following example shows the effect of the <code>ByVal</code> keyword used to passed an Integer to an external routine:</p> <pre> Declare Sub Foo Lib "MyLib" (ByRef i As Integer) i% = 6 Foo ByVal i% 'Pass a 2-byte Integer. Foo i% 'Pass a 4-byte pointer to an Integer. </pre> <p>Since the Foo routine expects to receive a pointer to an Integer, the first call to Foo will have unpredictable results.</p>
Exam- ple	<p>This example demonstrates the use of the <code>ByVal</code> keyword.</p> <pre> Sub Foo(a As Integer) a = a + 1 End Sub Sub Main() Dim i As Integer i = 10 Foo i MsgBox "The ByVal value is: " & i 'Displays 11 (Foo changed the value). Foo ByVal i MsgBox "The ByVal value is still: " & i 'Displays 11 (Foo did not change the value). End Sub </pre>
See Also	<p>() (on page 297) (keyword), ByRef (on page 352) (keyword).</p>

C

C

Call (statement)
CancelButton (statement)
CBool (function)
CCur (function)
CDate, CDate (functions)
Cdbl (function)
ChDir (statement)
ChDrive (statement)
CheckBox (statement)
Choose (function)
Chr, Chr\$, ChrB, ChrB\$, ChrW, ChrW\$ (functions)
CInt (function)
Clipboard\$ (function)
Clipboard\$ (statement)
Clipboard.Clear (method)
Clipboard.GetFormat (method)
Clipboard.GetText (method)
Clipboard.SetText (method)
CLng (function)
Close (statement)
ComboBox (statement)
Command, Command\$ (function)
Comments (topic)
Comparison Operators (topic)
Const (statement)
Constants (topic)
Cos (function)

CreateObject (function)
CSng (function)
CStr (function)
CurDir, CurDir\$ (function)
Currency (data type)
CVar (function)
CVErr (function)

Call (statement)

Syn- tax	<code>Call subroutine_name [(arguments)]</code>
De- scrip- tion	Transfers control to the given subroutine, optionally passing the specified arguments.
Com- ments	Using this statement is equivalent to: <code>subroutine_name [arguments]</code> Use of the Call statement is optional. The Call statement can only be used to execute subroutines; functions cannot be executed with this statement. The subroutine to which control is transferred by the Call statement must be declared outside of the Main procedure, as shown in the following example.
Exam- ple	<p>This example demonstrates the use of the Call statement to pass control to another function.</p> <pre> Sub Example_Call(s\$) 'This subroutine is declared externally to Main and displays the text 'passed in the parameter s\$. MsgBox "Call: " & s\$ End Sub Sub Main() 'This example assigns a string variable to display, then calls subroutine 'Example_Call, passing parameter S\$ to be displayed in a message box 'within the subroutine. s\$ = "DAVE" Example_Call s\$ Call Example_Call("SUSAN") End Sub </pre>

See	Goto (on page 544) (statement); GoSub (on page 543) (statement); Declare (on page 412)
Also	(statement).

CDBl (function)

Syn- tax	<code>CDBl</code> (expression)
De- scrip- tion	Converts any expression to a Double .
Com- ments	This function accepts any expression convertible to a Double , including strings. A runtime error is generated if expression is Null . Empty is treated as 0.0 . When passed a numeric expression, this function has the same effect as assigning the numeric expression number to a Double . When used with variants, this function guarantees that the variant will be assigned a Double (VarType 5).
Exam- ple	<p>This example displays the result of two numbers as a Double.</p> <pre> Sub Main() i% = 100 j! = 123.44 MsgBox "The double value is: " & CDBl(i% * j!) End Sub </pre>
See Also	CCur (on page 357) (function); CBool (on page 356) (function); CDate, CVDate (on page 358) (functions); CInt (on page 364) (function); CLng (on page 372) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVErr (on page 389) (function); Double (on page 458) (data type).

CBool (function)

Syn- tax	CBool (expression)
De- scrip- tion	Converts expression to True or False , returning a Boolean value.

Comments	<p>The expression parameter is any expression that can be converted to a Boolean . A runtime error is generated if expression is Null . All numeric data types are convertible to Boolean . If expression is zero, then the CBool returns False ; otherwise, CBool returns True . Empty is treated as False . If expression is a String , then CBool first attempts to convert it to a number, then converts the number to a Boolean . A runtime error is generated if expression cannot be converted to a number. A runtime error is generated if expression cannot be converted to a Boolean .</p>
Example	<p>This example uses CBool to determine whether a string is numeric or just plain text.</p> <pre data-bbox="305 594 1419 1050"> Sub Main() Dim IsNumericOrDate As Boolean s\$ = 34224.54 IsNumeric = CBool(IsNumeric(s\$)) If IsNumeric = True Then MsgBox s\$ & " is either a valid number!" Else MsgBox s\$ & " is not a valid number!" End If End Sub </pre>
See Also	<p>CCur (on page 357) (function); CDate, CVDate (on page 358) (functions); Cdbl (on page 356) (function); CInt (on page 364) (function); CLng (on page 372) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVer (on page 389) (function); Boolean (on page 351) (data type).</p>

CCur (function)

Syntax	CCur (expression)
Description	Converts any expression to a Currency .
Comments	<p>This function accepts any expression convertible to a Currency , including strings. A runtime error is generated if expression is Null or a String not convertible to a number. Empty is treated as 0. When passed a numeric expression, this function has the same effect as assigning the numeric expression number to a Currency . When used with variants, this function guarantees that the variant will be assigned a Currency (VarType 6).</p>

Example	<p>This example displays the value of a String converted into a Currency value.</p> <pre> Sub Main() i\$ = "100.44" MsgBox "The currency value is: " & CCur(i\$) End Sub </pre>
See Also	<p>CBool (on page 356) (function); CDate, CVDate (on page 358) (functions); Cdbl (on page 356) (function); CInt (on page 364) (function); CLng (on page 372) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVErr (on page 389) (function); Currency (on page 387) (data type).</p>

CDate, CVDate (functions)

Syntax	<p>CDate (expression) CVDate (expression)</p>
Description	<p>Converts expression to a date, returning a Date value.</p>
Comments	<p>The expression parameter is any expression that can be converted to a Date. A runtime error is generated if expression is Null. If expression is a String, an attempt is made to convert it to a Date using the current country settings. If expression does not represent a valid date, then an attempt is made to convert expression to a number. A runtime error is generated if expression cannot be represented as a date. These functions are sensitive to the date and time formats of your computer. The CDate and CVDate functions are identical.</p>
Example	<p>This example takes two dates and computes the difference between them.</p> <pre> Sub Main() Dim date1 As Date Dim date2 As Date Dim diff As Date date1 = CDate("#1/1/1994#") date2 = CDate("February 1, 1994") diff = DateDiff("d",date1,date2) MsgBox "The date difference is " & CInt(diff) & " days." End Sub </pre>

See Also	CCur (on page 357) (function); CBool (on page 356) (function); CDBl (on page 356) (function); CInt (on page 364) (function); CLng (on page 372) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVer (on page 389) (function); Date (on page 392) (data type).
----------	---

ChDir (statement)

Syntax	ChDir newdir\$
Description	Changes the current directory of the specified drive to newdir\$. This routine will not change the current drive. (See ChDrive [statement].)
Example	<p>This example saves the current directory, then changes to the root directory, displays the old and new directories, restores the old directory, and displays it.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() save\$ = CurDir\$ ChDir(Basic.PathSeparator\$) MsgBox "Old directory: " & save\$ & crlf & "New directory: " & CurDir\$ ChDir(save\$) MsgBox "Directory restored to: " & CurDir\$ End Sub </pre>
See Also	ChDrive (on page 359) (statement); CurDir, CurDir\$ (on page 387) (functions); Dir, Dir\$ (on page 427) (functions); MkDir (on page 608) (statement); Rmdir (on page 694) (statement).

ChDrive (statement)

Syntax	ChDrive DriveLetter\$
Description	Changes the default drive to the specified drive.
Comments	Only the first character of DriveLetter\$ is used. DriveLetter\$ is not case-sensitive. If DriveLetter\$ is empty, then the current drive is not changed.

Example	<p>This example allows the user to select a new current drive and uses ChDrive to make their choice the new current drive.</p> <pre> Const crlf\$ = Chr\$(13) + Chr\$(10) Sub Main() Dim d() old\$ = FileParse\$(CurDir,1) DiskDrives d Again: r = SelectBox("Available Drives","Select new current drive:",d) On Error Goto Error_Trap If r <> -1 Then ChDrive d@ MsgBox "Old Current Drive: " & old\$ & crlf & "New Current Drive: " & CurDir End Error_Trap: MsgBox Error(err) Resume Again End Sub </pre>
See Also	<p>ChDir (on page 359) (statement); CurDir, CurDir\$ (on page 387) (functions); Dir, Dir\$ (on page 427) (functions); Mkdir (on page 608) (statement); Rmdir (on page 694) (statement); DiskDrives (on page 428) (statement).</p>

CheckBox (statement)

Syntax	CheckBox X, Y, width, height, title\$, .Identifier				
Description	Defines a check box within a dialog box template.				
Comments	Check box controls are either on or off, depending on the value of .Identifier. This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements). The CheckBox statement requires the following parameters:				
	<table border="1"> <thead> <tr> <th data-bbox="295 1709 376 1785">Parameter</th> <th data-bbox="383 1709 1419 1785">Description</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Parameter	Description		
Parameter	Description				

	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	Width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Title\$	String containing the text that appears within the check box. This text may contain an ampersand character to denote an accelerator letter, such as "&Font" for Font (indicating that the Font control may be selected by pressing the F accelerator key).
	Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the state of the check box (1 = checked; 0 = unchecked). This variable can be accessed using the syntax: DialogVariable.Identifier.
		When the dialog box is first created, the value referenced by .Identifier is used to set the initial state of the check box. When the dialog box is dismissed, the final state of the check box is placed into this variable. By default, the .Identifier variable contains 0, meaning that the check box is unchecked.
Example		<p>This example displays a dialog box with two check boxes in different states.</p> <pre> Sub Main() Begin Dialog SaveOptionsTemplate 36,32,151,52,"Save" GroupBox 4,4,84,40,"GroupBox" CheckBox 12,16,67,8,"Include heading",.IncludeHeading CheckBox 12,28,73,8,"Expand keywords",.ExpandKeywords OKButton 104,8,40,14,.OK CancelButton 104,28,40,14,.Cancel End Dialog Dim SaveOptions As SaveOptionsTemplate SaveOptions.IncludeHeading = 1 'Check box initially on. SaveOptions.ExpandKeywords = 0 'Check box initially off. r% = Dialog(SaveOptions) If r% = -1 Then MsgBox "OK was pressed." End If End Sub </pre>
See Also		CancelButton (on page 365) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (state-

	ment); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin Dialog (on page 348) (statement), PictureButton (on page 671) (statement).
Notes	Accelerators are underlined, and the accelerator combination Alt+letter is used.

Choose (function)

Syn-tax	<code>Choose(index,expression1,expression2,...,expression13)</code>
De-scrip-tion	Returns the expression at the specified index position.
Com-ments	The index parameter specifies which expression is to be returned. If index is 1, then expression1 is returned; if index is 2, then expression2 is returned, and so on. If index is less than 1 or greater than the number of supplied expressions, then Null is returned. The Choose function returns the expression without converting its type. Each expression is evaluated before returning the selected one.
Exam-ple	<p>This example assigns a variable of indeterminate type to a.</p> <pre> Sub Main() Dim a As Variant Dim c As Integer c% = 2 a = Choose(c%,"Hello, world",#1/1/94#,5.5,False) MsgBox "Item " & c% & " is '" & a & "'" 'Displays the date passed as parameter 2. End Sub </pre>
See Also	Switch (on page 744) (function); IIf (on page 555) (function); If...Then...Else (on page 553) (statement); Select...Case (on page 708) (statement).

Chr, Chr\$, ChrB, ChrB\$, ChrW, ChrW\$ (functions)

Syn-tax	<code>Chr[\$](charcode) ChrB[\$](charcode) ChrW[\$](charcode)</code>
---------	--

De- scrip- tion	Returns the character whose value is Code.			
Com- ments	The <code>Chr\$</code> , <code>ChrB\$</code> , and <code>ChrW\$</code> functions return a String , whereas the <code>Chr</code> , <code>ChrB</code> , and <code>ChrW</code> functions return a String variant. These functions behave differently depending on the string format used by BasicScript. These differences are summarized in the following table:			
	Func- tion	String For- mat	Value between	Returns a
	<code>Chr[\$]</code>	SBCS	0 and 255	1-byte character string.
		MBCS	-32768 and 32767	1-byte or 2-byte MBCS character string depending on charcode.
		Wide	-32768 and 32767	2-byte character string.
	<code>ChrB[\$]</code>	SBCS	0 and 255	1-byte character string.
		MBCS	0 and 255	1-byte character string.
		Wide	0 and 255	1-byte character string.
	<code>ChrW[\$]</code>	SBCS	0 and 255	1-byte character string (same as the <code>Chr</code> and <code>Chr\$</code> func- tions)
		MBCS	-32768 and 32767	1-byte or 2-byte MBCS character string depending on charcode.
		Wide	-32768 and 32767	2-byte character string.
	The <code>Chr\$</code> function can be used within constant declarations, as in the following example: <code>Const crlf = Chr\$(13) + Chr\$(10)</code> Some common uses of this function are:			
	<code>Chr\$(9)</code>	Tab		
	<code>Chr\$(13) + Chr\$(10)</code>	End-of-line (carriage return, linefeed)		
	<code>Chr\$(26)</code>	End-of-file		
	<code>Chr\$(0)</code>	Null		
Exam- ple	<code>Sub Main()</code>			

	<pre> 'Concatenates carriage return (13) and line feed (10) to 'CRLF\$, then displays a multiple-line message using CRLF\$ 'to separate lines. CrLf\$ = Chr\$(13) + Chr\$(10) MsgBox "First line." &CrLf\$ & "Second line." 'Fills an array with the ASCII characters for ABC and 'displays their corresponding characters. Dim a%(2) For i = 0 To 2 a%(i) = (65 + i) Next i MsgBox "The first three elements of the array are: " _ & Chr\$(a%(0)) & Chr\$(a%(1)) & Chr\$(a%(2)) End Sub </pre>
See Also	Asc, AscB, AscW (on page 332) (functions); Str, Str\$ (on page 737) (functions).

CInt (function)

Syntax	<code>CInt(expression)</code>
Description	Converts expression to an Integer .
Comments	<p>This function accepts any expression convertible to an Integer, including strings. A runtime error is generated if expression is Null. Empty is treated as 0. The passed numeric expression must be within the valid range for integers:</p> <pre style="background-color: #f0f0f0; padding: 5px;">-32768 <= expression <= 32767</pre> <p>A runtime error results if the passed expression is not within the above range. When passed a numeric expression, this function has the same effect as assigning a numeric expression to an</p>

	Integer . Note that integer variables are rounded before conversion. When used with variants, this function guarantees that the expression is converted to an Integer variant (VarType 2).
Example	<p>This example demonstrates the various results of integer manipulation with CInt.</p> <pre> Sub Main() '(1) Assigns i# to 100.55 and displays its integer representation (101). I# = 100.55 MsgBox "The value of CInt(i) = " & CInt(i#) '(2) Sets j# to 100.22 and displays the CInt representation (100). j# = 100.22 MsgBox "The value of CInt(j) = " & CInt(j#) '(3) Assigns k% (integer) to the CInt sum of j# and k% and displays k% '(201). k% = CInt(i# + j#) MsgBox "The integer sum of 100.55 and 100.22 is: " & k% '(4) Reassigns i# to 50.35 and recalculates k%, then displays the result '(note rounding). i# = 50.35 k% = CInt(i# + j#) MsgBox "The integer sum of 50.35 and 100.22 is: " & k% End Sub </pre>
See Also	CCur (on page 357) (function); CBool (on page 356) (function); CDate, CVDate (on page 358) (functions); CDBl (on page 356) (function); CLng (on page 372) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVErr (on page 389) (function); Integer (on page 566) (data type).

CancelButton (statement)

Syntax	CancelButton X, Y, width, height [,Identifier]
Description	Defines a Cancel button that appears within a dialog box template.
Comments	This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements). Selecting the Cancel button (or pressing Esc) dismisses the user dialog box, causing the Dialog function to return 0 . (Note: A dialog function can redefine this behavior.) Pressing the Esc key or double-clicking the close box will have no effect if a dialog

	box does not contain a CancelButton statement. The CancelButton statement requires the following parameters:	
	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Identifier	Optional parameter specifying the name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the word Cancel is used.
	A dialog box must contain at least one OKButton , CancelButton , or PushButton statement; otherwise, the dialog box cannot be dismissed.	
Example	<p>This example creates a sample dialog box with OK and Cancel buttons.</p> <pre> Sub Main() Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit" Text 4,8,108,8,"Are you sure you want to exit?" CheckBox 32,24,63,8,"Save Changes",.SaveChanges OKButton 12,40,40,14 CancelButton 60,40,40,14 End Dialog Dim QuitDialog As QuitDialogTemplate rc% = Dialog(QuitDialog) Select Case rc% Case -1 MsgBox "OK was pressed!" Case 1 MsgBox "Cancel was pressed!" End Select End Sub </pre>	
See Also	CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653)	

(statement); [Picture \(on page 657\)](#) (statement); [PushButton \(on page 671\)](#) (statement); [Text \(on page 752\)](#) (statement); [TextBox \(on page 753\)](#) (statement); [Begin \(on page 348\) Dialog \(on page 348\)](#) (statement), [PictureBox \(on page 659\)](#) (statement).

Clipboard\$ (function)

Syntax	Clipboard\$()
Description	Returns a String containing the contents of the Clipboard.
Comments	If the Clipboard doesn't contain text or the Clipboard is empty, then a zero-length string is returned.
Example	<p>This example puts text on the Clipboard, displays it, clears the Clipboard, and displays the Clipboard again.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Clipboard\$ "Hello out there!" MsgBox "The text in the Clipboard is:" & crlf & Clipboard\$ Clipboard.Clear MsgBox "The text in the Clipboard is:" & crlf & Clipboard\$ End Sub </pre>
See Also	Clipboard\$ (on page 367) (statement); Clipboard.GetText (on page 371) (method); Clipboard.SetText (on page 371) (method).

Clipboard \$ (statement)

Syntax	Clipboard\$ NewContent\$
Description	Copies NewContent\$ into the Clipboard.
Example	<p>This example puts text on the Clipboard, displays it, clears the Clipboard, and displays the Clipboard again.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Clipboard\$ "Hello out there!" MsgBox "The text in the Clipboard is:" & crlf & Clipboard\$ </pre>

	<pre>Clipboard.Clear MsgBox "The text in the Clipboard is now:" & crlf & Clipboard\$ End Sub</pre>
See Also	Clipboard\$ (on page 367) (function); Clipboard.GetText (on page 371) (method); Clipboard.SetText (on page 371) (method).

Clipboard.Clear (method)

Syntax	Clipboard.Clear
Description	This method clears the Clipboard by removing any content.
Example	<p>This example puts text on the Clipboard, displays it, clears the Clipboard, and displays the Clipboard again.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Clipboard\$ "Hello out there!" MsgBox "The text in the Clipboard before clearing:" & crlf & Clipboard\$ Clipboard.Clear MsgBox "The text in the Clipboard after clearing:" & crlf & Clipboard\$ End Sub</pre>

CreateObject (function)

Syntax	CreateObject (class\$)
Description	Creates an OLE automation object and returns a reference to that object.
Comments	The class\$ parameter specifies the application used to create the object and the type of object being created. It uses the following syntax: "application.class", where application is the application used to create the object and class is the type of the object to create. At runtime, CreateObject looks for the given application and runs that application if found. Once the object is created, its properties and methods can be accessed using the dot syntax (e.g., object.property = value). There may be a slight delay when an automation server is loaded (this depends on the

speed with which a server can be loaded from disk). This delay is reduced if an instance of the automation server is already loaded.

**Exam-
ples** This first example instantiates Microsoft Excel. It then uses the resulting object to make Excel visible and then close Excel.

```
Sub Main()

  Dim Excel As Object

  On Error GoTo Trap1          'Set error trap.

  Set Excel = CreateObject("excel.application") 'Instantiate object.

  Excel.Visible = True        'Make Excel visible.

  Sleep 5000                   'Wait 5 seconds.

  Excel.Quit                   'Close Excel.

  Exit Sub                     'Exit before error trap.

Trap1:

  MsgBox "Can't create Excel object." 'Display error message.

  Exit Sub                     'Reset error handler.

End Sub
```

This second example uses CreateObject to instantiate a Visio object. It then uses the resulting object to create a new document.

```
Sub Main()

  Dim Visio As Object

  Dim doc As Object

  Dim page As Object

  Dim shape As Object

  On Error Goto NO_VISIO

  Set Visio = CreateObject("visio.application") 'Create Visio object.

  On Error Goto 0

  Set doc = Visio.Documents.Add("") 'Create a new document.

  Set page = doc.Pages(1)          'Get first page.

  Set shape = page.DrawRectangle(1,1,4,4) 'Create a new shape.

  shape.text = "Hello, world."     'Set text within shape.

  End

NO_VISIO:

  MsgBox "'Visio' cannot be found!", vbExclamation

End Sub
```

See	GetObject (on page 540) (function); Object (on page 633) (data type).
Also	

Clipboard.GetFormat (method)

Syn- tax	WhichFormat = Clipboard.GetFormat (format)	
De- scrip- tion	Returns TRUE if data of the specified format is available in the Clipboard; returns FALSE otherwise.	
Com- ments	This method is used to determine whether the data in the Clipboard is of a particular format. The format parameter is an Integer representing the format to be queried:	
	Format	Description
	1	Text
	2	Bitmap
	3	Metafile
	8	Device-independent bitmap (DIB)
	9	Color palette
Exam- ple	<p>This example checks to see whether there is any text on the Clipboard, if so, it searches the text for a string matching what the user entered.</p> <pre> Option Compare Text Sub Main() r\$ = InputBox("Enter a word to search for:", "Scan Clipboard") If Clipboard.GetFormat(1) Then If Instr(Clipboard.GetText(1), r) = 0 Then MsgBox "" & r & "" & " was not found in the clipboard." Else MsgBox "" & r & "" & " is definitely in the clipboard." End If Else MsgBox "The Clipboard does not contain any text." End If End Sub </pre>	

See Also	Clipboard\$ (on page 367) (function); Clipboard\$ (on page 367) (statement).
----------	--

Clipboard.GetText (method)

Syntax	Text\$ = Clipboard.GetText ([format])
Description	Returns the text contained in the Clipboard.
Comments	The format parameter, if specified, must be 1 .
Example	<p>This example checks to see whether there is any text on the Clipboard, if so, it searches the text for a string matching what the user entered.</p> <pre> Option Compare Text Sub Main() r\$ = InputBox("Enter a word to search for:", "Scan Clipboard") If Clipboard.GetFormat(1) Then If Instr(Clipboard.GetText(1), r) = 0 Then MsgBox "" & r & "" & " was not found in the clipboard." Else MsgBox "" & r & "" & " is definitely in the clipboard." End If Else MsgBox "The Clipboard does not contain any text." End If End Sub </pre>
See Also	Clipboard\$ (on page 367) (statement); Clipboard\$ (on page 367) (function); Clipboard.SetText (on page 371) (method).

Clipboard.SetText (method)

Syntax	Clipboard.SetText data\$ [,format]
Description	Copies the specified text string to the Clipboard.

Comments	The data\$ parameter specifies the text to be copied to the Clipboard. The format parameter, if specified, must be 1 .
Example	<p>This example gets the contents of the Clipboard and uppercases it.</p> <pre> Sub Main() If Not Clipboard.GetFormat(1) Then Exit Sub Clipboard.SetText UCase(Clipboard.GetText(1)),1 End Sub </pre>
See Also	Clipboard\$ (on page 367) (statement); Clipboard.GetText (on page 371) (method); Clipboard\$ (on page 367) (function).

CLng (function)

Syntax	CLng (expression)
Description	Converts expression to a Long .
Comments	<p>This function accepts any expression convertible to a Long , including strings. A runtime error is generated if expression is Null . Empty is treated as 0 . The passed expression must be within the following range:</p> <pre> -2147483648 <= expression <= 2147483647 </pre> <p>A runtime error results if the passed expression is not within the above range. When passed a numeric expression, this function has the same effect as assigning the numeric expression to a Long . Note that long variables are rounded before conversion. When used with variants, this function guarantees that the expression is converted to a Long variant (VarType 3).</p>
Example	<p>This example displays the results for various conversions of i and j (note rounding).</p> <pre> Sub Main() I% = 100 j& = 123.666 MsgBox "The result of i * j is: " & CLng(i% * j&) 'Displays 12367. MsgBox "The new variant type of i is: " & Vartype(CLng(i%)) End Sub </pre>

See Also	CCur (on page 357) (function); CBool (on page 356) (function); CDate, CVDate (on page 358) (functions); CDBl (on page 356) (function); Clnt (on page 364) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVer (on page 389) (function); Long (on page 598) (data type).
----------	---

Close (statement)

Syntax	Close [[#] filename [, [#] filename]...]
Description	Closes the specified files.
Comments	If no arguments are specified, then all files are closed.
Example	<p>This example opens four files and closes them in various combinations.</p> <pre> Sub Main() Open "test1" For Output As #1 Open "test2" For Output As #2 Open "test3" For Random As #3 Open "test4" For Binary As #4 MsgBox "The next available file number is: " & FreeFile() Close #1 'Closes file 1 only. Close #2,#3 'Closes files 2 and 3. Close 'Closes all remaining files(4). MsgBox "The next available file number is: " & FreeFile() End Sub </pre>
See Also	Open (on page 642) (statement); Reset (on page 690) (statement); End (on page 487) (statement).

ComboBox (statement)

Syntax	ComboBox X,Y,width,height,ArrayVariable,.Identifier
Description	This statement defines a combo box within a dialog box template.

Comments	When the dialog box is invoked, the combo box will be filled with the elements from the specified array variable. This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements). The ComboBox statement requires the following parameters:	
	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Array-Variable	Single-dimensional array used to initialize the elements of the combo box. If this array has no dimensions, then the combo box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
	Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates a string variable whose value corresponds to the content of the edit field of the combo box. This variable can be accessed using the syntax: DialogVariable.Identifier
	When the dialog box is invoked, the elements from ArrayVariable are placed into the combo box. The .Identifier variable defines the initial content of the edit field of the combo box. When the dialog box is dismissed, the .Identifier variable is updated to contain the current value of the edit field.	
Example	<p>This example creates a dialog box that allows the user to select a day of the week.</p> <pre data-bbox="305 1423 1425 1871"> Sub Main() Dim days\$(6) days\$(0) = "Monday" days\$(1) = "Tuesday" days\$(2) = "Wednesday" days\$(3) = "Thursday" days\$(4) = "Friday" days\$(5) = "Saturday" days\$(6) = "Sunday" </pre>	

	<pre> Begin Dialog DaysDialogTemplate 16,32,124,96,"Days" OKButton 76,8,40,14,..OK Text 8,10,39,8,"&Weekdays:" ComboBox 8,20,60,72,days\$, .Days End Dialog Dim DaysDialog As DaysDialogTemplate DaysDialog.Days = Format(Now,"dddd") 'Set to today. r% = Dialog(DaysDialog) MsgBox "You selected: " & DaysDialog.Days End Sub </pre>
See Also	<p>CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin (on page 348) Dialog (on page 348) (statement), PictureButton (on page 659) (statement).</p>

Command, Command\$ (functions)

Syntax	Command[\$][()]
Description	Returns the argument from the command line used to start the application.
Comments	Command\$ returns a string, whereas Command returns a String variant.
Example	<p>This example checks to see if any command line parameters were used. If parameters were used they are displayed and a check is made to see if the user used the "/s" switch.</p> <pre> Sub Main() cmd\$ = Command If cmd\$ <> "" Then If (Instr(cmd\$,"/s")) <> 0 Then MsgBox "Safety Mode On!" Else </pre>


```

MsgBox "Safety Mode Off!"

End If

MsgBox "The command line startup options were: " & cmd$

Else

MsgBox "No command line startup options were used!"

End If

End Sub
    
```

See Also [Environ, Environ\\$ \(on page 488\)](#) (functions).

Comparison Operators (topic)

Syntax	Expression1 [< > <= >= <> =] expression2		
Description	Comparison operators return True or False depending on the operator.		
Comments	The comparison operators are listed in the following table:		
	Operator	Returns True If	
	>	expression1 is greater than expression2	
	<	expression1 is less than expression2	
	<=	expression1 is less than or equal to expression2	
	>=	expression1 is greater than or equal to expression2	
	<>	expression1 is not equal to expression2	
	=	expression1 is equal to expression2	
	This operator behaves differently depending on the types of the expressions, as shown in the following table:		
	If one expression is	and the other expression is	Then
	Numeric	Numeric	A numeric comparison is performed (see below).

	String	String	A string comparison is performed (see below).
	Numeric	String	A compile error is generated.
	Variant	String	A string comparison is performed (see below).
	Variant	Numeric	A variant comparison is performed (see below).
	Null variant	Any data type	Returns Null .
	Variant	Variant	A variant comparison is performed (see below).
<p>String Comparisons If the two expressions are strings, then the operator performs a text comparison between the two string expressions, returning True if expression1 is less than expression2. The text comparison is case-sensitive if Option Compare is Binary; otherwise, the comparison is case-insensitive. When comparing letters with regard to case, lowercase characters in a string sort greater than uppercase characters, so a comparison of "a" and "A" would indicate that "a" is greater than "A". Numeric Comparisons When comparing two numeric expressions, the less precise expression is converted to be the same type as the more precise expression. Dates are compared as doubles. This may produce unexpected results as it is possible to have two dates that, when viewed as text, display as the same date when, in fact, they are different. This can be seen in the following example:</p>			
<pre> Sub Main() Dim date1 As Date Dim date2 As Date date1 = Now date2 = date1 + 0.000001 'Adds a fraction of a second. MsgBox date2 = date1 'Prints False (the dates are different). MsgBox date1 & ", " & date2 'Prints two dates that are the same. End Sub </pre>			
<p>Variant Comparisons When comparing variants, the actual operation performed is determined at execution time according to the following table:</p>			
	If one variant is	and the other variant is	Then
	Numeric	Numeric	The variants are compared as numbers.
	String	String	The variants are compared as text.
	Numeric	String	The number is less than the string.
	Null	Any other data type	Null

	Numeric	Empty	The number is compared with 0.
	String	Empty	The string is compared with a zero-length string.
Example	<pre> Sub Main() 'Tests two literals and displays the result. If 5 < 2 Then MsgBox "5 is less than 2." Else MsgBox "5 is not less than 2." End If 'Tests two strings and displays the result. If "This" < "That" Then MsgBox "'This' is less than 'That'." Else MsgBox "'That' is less than 'This'." End If End Sub </pre>		
See Also	Operator Precedence (on page 648) (topic); Is (on page 570) (operator); Like (on page 587) (operator); Option Compare (on page 649) (statement).		

Const (statement)

Syntax	Const name [As type] = expression [,name [As type] = expression]...
Description	Declares a constant for use within the current script.
Comments	The name is only valid within the current Basic Control Engine script. Constant names must follow these rules: 1. Must begin with a letter. 2. May contain only letters, digits, and the underscore character. 3. Must not exceed 80 characters in length. 4. Cannot be a reserved word. Constant names are not case-sensitive.
	<p>The expression must be assembled from literals or other constants. Calls to functions are not allowed except calls to the Chr\$ function, as shown below:</p> <pre> Const s\$ = "Hello, there" + Chr(44) </pre>

Constants can be given an explicit type by declaring the name with a type-declaration character, as shown below:

```
Const a% = 5           'Constant Integer whose value is 5
Const b# = 5           'Constant Double whose value is 5.0
Const c$ = "5"         'Constant String whose value is "5"
Const d! = 5           'Constant Single whose value is 5.0
Const e& = 5           'Constant Long whose value is 5
```

The type can also be given by specifying the **As** type clause:

```
Const a As Integer = 5   'Constant Integer whose value is 5
Const b As Double = 5    'Constant Double whose value is 5.0
Const c As String = "5"  'Constant String whose value is "5"
Const d As Single = 5    'Constant Single whose value is 5.0
Const e As Long = 5      'Constant Long whose value is 5
```

You cannot specify both a type-declaration character and the type:

```
Const a% As Integer = 5  'THIS IS ILLEGAL.
```

If an explicit type is not given, then the Basic Control Engine script will choose the most imprecise type that completely represents the data, as shown below:

```
Const a = 5             'Integer constant
Const b = 5.5           'Single constant
Const c = 5.5E200       'Double constant
```

Constants defined within a **Sub** or **Function** are local to that subroutine or function. Constants defined outside of all subroutines and function can be used anywhere within that script. The following example demonstrates the scoping of constants:

```
Const DefFile = "default.txt"

Sub Test1
    Const DefFile = "foobar.txt"
    MsgBox DefFile      'Displays "foobar.txt".
End Sub

Sub Test2
    MsgBox DefFile      'Displays "default.txt".
End Sub
```

Example

This example displays the declared constants in a dialog box (crLf produces a new line in the dialog box).

	<pre> Const crlf = Chr\$(13) + Chr\$(10) Const greeting As String = "Hello, " Const question1 As String = "How are you today?" Sub Main() r = InputBox("Please enter your name", "Enter Name") MsgBox greeting & r & crlf & crlf & question1 End Sub </pre>
See Also	DefType (on page 421) (statement); Let (on page 586) (statement); = (statement); Constants (on page 380) (topic).

Constants (topic)

Constants are variables that cannot change value during script execution. The following constants are predefined by the Basic Control Engine:

Constant	Value	Description
ebMinimized	1	The application is minimized.
ebMaximized	2	The application is maximized.
ebRestored	3	The application is restored.
True	1	Boolean value True.
False	0	Boolean value False.
Empty	Empty	Variant of type 0, indicating that the variant is un-initialized.
Nothing	0	Value indicating that an object variable no longer references a valid object.
Null	Null	Variant of type 1, indicating that the variant contains no data.
ebCFText	1	Text.
ebCFBitmap	2	Bitmap
ebCFMetafile	3	Metafile.
ebCFDIB	8	Device-independent bitmap.
ebCFPalette	9	Palette
ebCFUnicode	13	Unicode text

ebUseSunday	0	Use the date setting as specified by the current locale.
ebSunday	1	Sunday.
ebMonday	2	Monday
ebTuesday	3	Tuesday
ebWednesday	4	Wednesday.
ebThursday	5	Thursday
ebFriday	6	Friday
ebSaturday	7	Saturday.
ebFirstJan1	1	Start with week in which January 1 occurs.
ebFirstFourDays	2	Start with first week with at least four days in the new year.
ebFirstFullWeek	3	Start with first full week of the year.
ebNormal	0	Read-only, archive, subdir, and none.
ebReadOnly	1	Read-only files.
ebHidden	2	Hidden files.
ebSystem	4	System files
ebVolume	8	Volume labels
ebDirectory	16	Subdirectory
ebArchive	32	Files that have changed since the last backup.
ebNone	64	Files with no attributes.
ebWindows		Windows executable file
ebRegular	1	Normal font (i.e., neither bold nor italic).
ebItalic	2	Italic font.
ebBold	4	Bold font.
ebBoldItalic	6	Bold-italic font.
ebIMENoOp	0	IME not installed
ebIMEOn	1	IME on
ebIMEOff	2	IME off

ebIMEDisabled	3	IME disabled
ebIMEHiragana	4	Hiragana double-byte character.
ebIMEKatakanaDbl	5	Katakana double-byte characters.
ebIMEKatakanaSng	6	Katakana single-byte characters.
ebIMEAlphaDbl	7	Alphanumeric double-byte characters.
ebIMEAlphaSng	8	Alphanumeric single-byte characters.
PI	3.1415...	Value of PI.
ebOKOnly	0	Displays only the OK button.
ebOKCancel	1	Displays OK and Cancel buttons.
ebAbortRetryIgnore	2	Displays Abort, Retry, and Ignore buttons.
ebYesNoCancel	3	Displays Yes, No, and Cancel buttons.
ebYesNo	4	Displays Yes and No buttons.
ebRetryCancel	5	Displays Cancel and Retry buttons.
ebCritical	16	Displays the stop icon.
ebQuestion	32	Displays the question icon.
ebExclamation	48	Displays the exclamation icon.
ebInformation	64	Displays the information icon.
ebApplication-Modal	0	The current application is suspended until the dialog box is closed.
ebDefaultButton1	0	First button is the default button.
ebDefaultButton2	256	Second button is the default button.
ebDefaultButton3	512	Third button is the default button.
ebSystemModal	4096	All applications are suspended until the dialog box is closed.
ebOK	1	Returned from MsgBox indicating that OK was pressed.
ebCancel	2	Returned from MsgBox indicating that Cancel was pressed.
ebAbort	3	Returned from MsgBox indicating that Abort was pressed.
ebRetry	4	Returned from MsgBox indicating that Retry was pressed.

ebIgnore	5	Returned from MsgBox indicating that Ignore was pressed.
ebYes	6	Returned from MsgBox indicating that Yes was pressed.
ebNo	7	Returned from MsgBox indicating that No was pressed.
ebLandscape	1	Landscape paper orientation.
ebPortrait	2	Portrait paper orientation
ebLeftButton	1	Left mouse button
ebRightButton	2	Right mouse button
ebHide	0	Application is initially hidden.
ebNormalFocus	1	Application is displayed at the default position and has the focus.
ebMinimizedFocus	2	Application is initially minimized and has the focus.
ebMaximizedFocus	3	Application is maximized and has the focus.
ebNormalNoFocus	4	Application is displayed at the default position and does not have the focus.
ebMinimizedNoFocus	6	Application is minimized and does not have the focus.
ebUpperCase	1	Converts string to uppercase.
ebLowerCase	2	Converts string to lowercase.
ebProperCase	3	Capitalizes the first letter of each word.
ebWide	4	Converts narrow characters to wide characters.
ebNarrow	8	Converts wide characters to narrow characters.
ebKatakana	16	Converts Hiragana characters to Katakana characters.
ebHiragana	32	Converts Katakana characters to Hiragana characters.
ebUnicode	64	Converts string from MBCS to UNICODE.
ebFromUnicode	128	Converts string from UNICODE to MBCS.
ebEmpty	0	Variant has not been initialized.
ebNull	1	Variant contains no valid data.
ebInteger	2	Variant contains an Integer.

ebLong	3	Variant contains a Long.
ebSingle	4	Variant contains a Single.
ebDouble	5	Variant contains a Double.
ebCurrency	6	Variant contains a Currency.
ebDate	7	Variant contains a Date.
ebString	8	Variant contains a String.
ebObject	9	Variant contains an Object.
ebError	10	Variant contains an Error.
ebBoolean	11	Variant contains a Boolean.
ebVariant	12	Variant contains an array of Variants.
ebDataObject	13	Variant contains a data object.
ebArray	8192	Added to any of the other types to indicate an array of that type.
Constant	Value	Description
ebBack	Chr\$(8)	String containing a backspace.
ebCr	Chr\$(13)	String containing a carriage return.
ebCrLf	Chr\$(13) & Chr\$(10)	String containing a carriage-return linefeed pair.
ebFormFeed	Chr\$(11)	String containing a form feed.
ebLf	Chr\$(10)	String containing a line feed.
ebNullChar	Chr\$(0)	String containing a single null character.
ebNullString	0	Special string value used to pass null pointers to external routines.
ebTab	Chr\$(9)	String containing a tab.
ebVerticalTab	Chr\$(12)	String containing a vertical tab.
Constant	Value	
Win32	True if development environment is 32-bit Windows.	
Empty	Empty	

False	False
Null	Null
True	True
You can define your own constants using the <code>Const</code> statement. Preprocessor constants are defined using <code>#Const</code> .	

Cos (function)

Syntax	Cos (angle)
Description	Returns a Double representing the cosine of angle.
Comments	The angle parameter is a Double specifying an angle in radians.
Example	<p>This example assigns the cosine of pi/4 radians (45 degrees) to C# and displays its value.</p> <pre>Sub Main() c# = Cos(3.14159 / 4) MsgBox "The cosine of 45 degrees is: " & c# End Sub</pre>
See Also	Tan (on page 751) (function); Sin (on page 718) (function); Atn (on page 337) (function).

CSng (function)

Syntax	CSng (expression)
Description	Converts expression to a Single .
Comments	This function accepts any expression convertible to a Single , including strings. A runtime error is generated if expression is Null . Empty is treated as 0.0 . A runtime error results if the passed expression is not within the valid range for Single . When passed a numeric expression, this function has the same effect as assigning the numeric expression to a Single . When used

	with variants, this function guarantees that the expression is converted to a Single variant (VarType 4).
Example	<p>This example displays the value of a String converted to a Single.</p> <pre> Sub Main() s\$ = "100" MsgBox "The single value is: " & CSng(s\$) End Sub </pre>
See Also	CCur (on page 357) (function); CBool (on page 356) (function); CDate, CDate (on page 358) (functions); CDBl (on page 356) (function); CInt (on page 364) (function); CLng (on page 372) (function); CStr (on page 386) (function); CVar (on page 388) (function); CVer (on page 389) (function); Single (on page 718) (data type).

CStr (function)

Syntax	CStr (expression)												
Description	Converts expression to a String .												
Comments	Unlike Str\$ or Str , the string returned by CStr will not contain a leading space if the expression is positive. Further, the CStr function correctly recognizes thousands and decimal separators for your locale. Different data types are converted to String in accordance with the following rules:												
	<table border="1"> <thead> <tr> <th>Data Type</th> <th>CStr Returns</th> </tr> </thead> <tbody> <tr> <td>Any numeric type</td> <td>A string containing the number without the leading space for positive values.</td> </tr> <tr> <td>Date</td> <td>A string converted to a date using the short date format.</td> </tr> <tr> <td>Boolean</td> <td>A string containing either TRUE or FALSE.</td> </tr> <tr> <td>Null variant</td> <td>A runtime error.</td> </tr> <tr> <td>Empty variant</td> <td>A zero-length string.</td> </tr> </tbody> </table>	Data Type	CStr Returns	Any numeric type	A string containing the number without the leading space for positive values.	Date	A string converted to a date using the short date format.	Boolean	A string containing either TRUE or FALSE.	Null variant	A runtime error.	Empty variant	A zero-length string.
Data Type	CStr Returns												
Any numeric type	A string containing the number without the leading space for positive values.												
Date	A string converted to a date using the short date format.												
Boolean	A string containing either TRUE or FALSE.												
Null variant	A runtime error.												
Empty variant	A zero-length string.												
Example	<p>This example displays the value of a Double converted to a String.</p> <pre> Sub Main() s# = 123.456 </pre>												

	<pre>MsgBox "The string value is: " & CStr(s#) End Sub</pre>
See Also	CCur (on page 357) (function); CBool (on page 356) (function); CDate, CDate (on page 358) (functions); CDBl (on page 356) (function); CInt (on page 364) (function); CLng (on page 372) (function); CSng (on page 385) (function); CVar (on page 388) (function); CVer (on page 389) (function); String (on page 742) (data type); Str, Str\$ (on page 737) (functions).

CurDir, CurDir\$ (functions)

Syntax	CurDir[\$] [(drive\$)]
Description	Returns the current directory on the specified drive. If no drive\$ is specified or drive\$ is zero-length, then the current directory on the current drive is returned.
Comments	CurDir\$ returns a String , whereas CurDir returns a String variant. The script generates a runtime error if drive\$ is invalid.
Example	<p>This example saves the current directory, changes to the next higher directory, and displays the change; then restores the original directory and displays the change. Note: The dot designators will not work with all platforms.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() save\$ = CurDir ChDir (".") MsgBox "Old directory: " & save\$ & crlf & "New directory: " & CurDir ChDir (save\$) MsgBox "Directory restored to: " & CurDir End Sub</pre>
See Also	ChDir (on page 359) (statement); ChDrive (on page 359) (statement); Dir, Dir\$ (on page 427) (functions); MkDir (on page 608) (statement); Rmdir (on page 694) (statement).

Currency (data type)

Syn- tax	Currency
De- scrip- tion	A data type used to declare variables capable of holding fixed-point numbers with 15 digits to the left of the decimal point and 4 digits to the right.
Com- ments	<p>Currency variables are used to hold numbers within the following range:</p> <pre style="background-color: #f0f0f0; padding: 5px;">-922,337,203,685,477.5808 <= currency <= 922,337,203,685,477.5807</pre> <p>Due to their accuracy, Currency variables are useful within calculations involving money. The type-declaration character for Currency is @ . Storage Internally, currency values are 8-byte integers scaled by 10000. Thus, when appearing within a structure, currency values require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required.</p>
See Also	Date (on page 392) (data type); Double (on page 458) (data type); Integer (data type); Long (on page 598) (data type); Object (on page 633) (data type); Single (on page 718) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement); CCur (on page 357) (function).

CVar (function)

Syn- tax	CVar (expression)
De- scrip- tion	Converts expression to a Variant .
Com- ments	<p>This function is used to convert an expression into a variant. Use of this function is not necessary (except for code documentation purposes) because assignment to variant variables automatically performs the necessary conversion:</p> <pre style="background-color: #f0f0f0; padding: 5px;">Sub Main() Dim v As Variant v = 4 & "th" 'Assigns "4th" to v. MsgBox "You came in: " & v v = CVar(4 & "th") 'Assigns "4th" to v. MsgBox "You came in: " & v End Sub</pre>

Example	<p>This example converts an expression into a Variant.</p> <pre> Sub Main() Dim s As String Dim a As Variant s = CStr("The quick brown fox ") msg1 = CVar(s & "jumped over the lazy dog.") MsgBox msg1 End Sub </pre>
See Also	<p>CCur (on page 357) (function); CBool (on page 356) (function); CDate, CVDate (on page 358) (functions); Cdbl (on page 356) (function); Clnt (on page 364) (function); CLng (on page 372) (function); CSng (on page 385) (function); CStr (on page 386) (function); CVErr (on page 389) (function); Variant (on page 771) (data type).</p>

CVErr (function)

Syntax	CVErr (expression)
Description	Converts expression to an error.
Comments	<p>This function is used to convert an expression into a user-defined error number. A runtime error is generated under the following conditions: If expression is Null . If expression is a number outside the legal range for errors, which is as follows:</p> <pre> 0 <= expression <= 65535 </pre> <p>If expression is Boolean . If expression is a String that can't be converted to a number within the legal range. Empty is treated as 0.</p>
Example	<p>This example simulates a user-defined error and displays the error number.</p> <pre> Sub Main() MsgBox "The error is: " & CStr(CVErr(2046)) End Sub </pre>
See Also	<p>CCur (on page 357) (function); CBool (on page 356) (function); CDate, CVDate (on page 358) (functions); Cdbl (on page 356) (function); Clnt (on page 364) (function); CLng (on</p>

[page 372](#)) (function); [CSng \(on page 385\)](#) (function); [CStr \(on page 386\)](#) (function); [CVar \(on page 388\)](#) (function), [IsError \(on page 572\)](#) (function).

Comments (topic)

Comments can be added to Basic Control Engine script code in the following manner: All text between a single quotation mark and the end of the line is ignored:

```
MsgBox "Hello" 'Displays a message box.
```

The **REM** statement causes the compiler to ignore the entire line:

```
REM This is a comment.
```

The Basic Control Engine supports C-style multiline comment blocks `/*...*/`, as shown in the following example:

```
MsgBox "Before comment"

/* This stuff is all commented out.

This line, too, will be ignored.

This is the last line of the comment. */

MsgBox "After comment"
```

C-style comments can be nested.

D

D

Date (data type)
Date, Date\$ (functions)
Date, Date\$ (statements)
DateAdd (function)
DateDiff (function)
DatePart (function)
DateSerial (function)
DateValue (function)
Day (function)

DDB (function)
DDEExecute (statement)
DDEInitiate (function)
DDEPoke (statement)
DDERequest, DDERequest\$ (function)
DDESend (statement)
DDETerminate (statement)
DDETerminateAll (statement)
DDETimeout (statement)
Declare (statement)
DefType (statement)
DeleteSetting (statement)
Dialog (function)
Dialog (statement)
Dim (statement)
Dir, Dir\$ (function)
DiskDrives (statement)
DiskFree (function)
DlgCaption (function)
DlgCaption (statement)
DlgControlId (function)
DlgEnable (function)
DlgEnable (statement)
DlgFocus (function)
DlgFocus (statement)
DlgListBoxArray (function)
DlgListBoxArray (statement)

DlgProc (function)
DlgSetPicture (statement)
DlgText (statement)
DlgText\$ (function)
DlgValue (function)
DlgValue (statement)
DlgVisible (function)
DlgVisible (statement)
Do...Loop (statement)
DoEvents (function)
DoEvents (statement)
Double (data type)
DropListBox (statement)

Date (data type)

Syn- tax	Date
De- scrip- tion	A data type capable of holding date and time values.
Com- ments	<p>Date variables are used to hold dates within the following range:</p> <pre>January 1, 100 00:00:00 <= date <= December 31, 9999 23:59:59</pre> <pre>-6574340 <= date <= 2958465.99998843</pre> <p>Internally, dates are stored as 8-byte IEEE double values. The integer part holds the number of days since December 31, 1899, and the fractional part holds the number of seconds as a fraction of the day. For example, the number 32874.5 represents January 1, 1990 at 12:00:00. When appearing within a structure, dates require 8 bytes of storage. Similarly, when used with binary or random files, 8 bytes of storage are required. There is no type-declaration character for Date.</p>

	<p>Date variables that haven't been assigned are given an initial value of 0 (i.e., December 31, 1899).</p>
	<p>Date Literals Literal dates are specified using number signs, as shown below:</p> <pre>Dim d As Date d = #January 1, 1990#</pre> <p>The interpretation of the date string (i.e., January 1, 1990 in the above example) occurs at runtime, using the current country settings. This is a problem when interpreting dates such as 1/2/1990. If the date format is M/D/Y, then this date is January 2, 1990. If the date format is D/M/Y, then this date is February 1, 1990. To remove any ambiguity when interpreting dates, use the universal date format: <code>date_variable = #YY/MM/DD HH:MM:SS#</code> The following example specifies the date June 3, 1965 using the universal date format:</p> <pre>Dim d As Date d = #1965/6/3 10:23:45#</pre>
See Also	<p>Currency (on page 387) (data type); Double (on page 458) (data type); Integer (on page 566) (data type); Long (on page 598) (data type); Object (on page 633) (data type); Single (on page 718) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement); CDate, CVDate (on page 358) (functions).</p>

Date, Date\$ (functions)

Syn-tax	Date[\$] [()]
De-scrip-tion	Returns the current system date.
Com-ments	<p>The Date\$ function returns the date using the short date format. The Date function returns the date as a Date variant. Use the Date/Date\$ statements to set the system date. The date is returned using the current short date format (defined by the operating system).</p> <p>The Date\$ function does not properly support international formats. Use the Date function instead.</p>
Exam-ple	This example saves the current date to TheDate\$, then changes the date and displays the result. It then changes the date back to the saved date and displays the restored date.

	<pre> ' When run with non-US Regional or International settings, ' the two message boxes may display different dates. ' One set of International Date Formats which shows this is: ' Short Date Format: dd.M.yy (ex: 02.01.97 for 2 January 1997) ' Long Date Format: dddd, dd M, yyyy (Thursday, 02 January 1997) Sub Main() ' Save the current date TheDate\$ = Date ' Set the date to one that may confuse the library functions ' (month and day < 12) Date = "01/02/97" ' 1 Feb 1997 MsgBox(Format\$(Date\$, "dddd")) ' This may show 2 Jan MsgBox(Format\$(Date, "dddd")) ' This may show 1 Feb ' Restore the date Date = TheDate\$ End Sub </pre>
See Also	CDate, CVDate (on page 358) (functions); Time, Time\$ (on page 755) (functions); Date, Date\$ (on page 394) (statements); Now (on page 629) (function); Format, Format\$ (on page 525) (functions); DateSerial (on page 400) (function); DateValue (on page 401) (function).

Date, Date\$ (statements)

Syntax	Date[\$] = newdate
Description	Sets the system date to the specified date.
Comments	The Date\$ statement requires a string variable using one of the following formats: MM-DD-YYYY MM-DD-YY MM/DD/YYYY MM/DD/YY, Where MM is a two-digit month between 1 and 31, DD is a two-digit day between 1 and 31, and YYYY is a four-digit year between 1/1/100 and 12/31/9999. The Date statement converts any expression to a date, including string and numeric values. Unlike the Date\$ statement, Date recognizes many different date formats, including abbreviated and full month names and a variety of ordering options. If newdate contains a time

	component, it is accepted, but the time is not changed. An error occurs if newdate cannot be interpreted as a valid date.
Example	<p>This example saves the current date to Cdate\$, then changes the date and displays the result. It then changes the date back to the saved date and displays the result.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() TheDate\$ = Date Date = "01/01/95" MsgBox "Saved date is: " & TheDate\$ & crlf & "Changed date is: " & Date Date = TheDate\$ MsgBox "Restored date to: " & TheDate\$ End Sub </pre>
See Also	Date, Date\$ (on page 393) (functions); Time, Time\$ (on page 756) (statements).
Notes	If you do not have permission to change the date, runtime error 70 will be generated.

DateAdd (function)

Syntax	<code>DateAdd (interval\$, increment&, date)</code>	
Description	Returns a Date variant representing the sum of date and a specified number (increment) of time intervals (interval\$).	
Comments	This function adds a specified number (increment) of time intervals (interval\$) to the specified date (date). The following table describes the parameters to the DateAdd function:	
	Parameter	Description
	Interval\$	String expression indicating the time interval used in the addition.
	Increment	Integer indicating the number of time intervals you wish to add. Positive values result in dates in the future; negative values result in dates in the past.
	Date	Any expression convertible to a Date .

	The interval\$ parameter specifies what unit of time is to be added to the given date. It can be any of the following:
Time	Intervale
"y"	Day of the year
"yyyy"	Year
"d"	Day
"m"	Month
"q"	Quarter
"ww"	Week
"h"	Hour
"n"	Minute
"s"	Second
"w"	Weekday
	<p>To add days to a date, you may use either day, day of the year, or weekday, as they are all equivalent (" d ", " y ", " w "). The DateAdd function will never return an invalid date/time expression. The following example adds two months to December 31, 1992:</p> <pre>s# = DateAdd("m",2,"December 31,1992")</pre> <p>In this example, s is returned as the double-precision number equal to " February 28, 1993 ", not " February 31, 1993 ". A runtime error is generated if you try to subtract a time interval that is larger than the time value of the date.</p>
Example	<p>This example gets today's date using the Date\$ function; adds three years, two months, one week, and two days to it; and then displays the result in a dialog box.</p> <pre>Sub Main() Dim sdate\$ sdate\$ = Date\$ NewDate# = DateAdd("yyyy",4,sdate\$) NewDate# = DateAdd("m",3,NewDate#) NewDate# = DateAdd("ww",2,NewDate#) NewDate# = DateAdd("d",1,NewDate#) s\$ = "Four years, three months, two weeks, and one day from now will be: " s\$ = s\$ & Format(NewDate#,"long date") </pre>

	<pre>MsgBox s\$ End Sub</pre>
See Also	DateDiff (on page 397) (function).

DateDiff (function)

Syn-tax	DateDiff (interval\$,date1,date2)	
Description	Returns a Date variant representing the number of given time intervals between date1 and date2.	
Comments	The following table describes the parameters:	
	Parameter	Description
	Interval\$	String expression indicating the specific time interval you wish to find the difference between.
	Date1	Any expression convertible to a Date . An example of a valid date/time string would be " January 1, 1994 " .
	Date2	Any expression convertible to a Date . An example of a valid date/time string would be " January 1, 1994 " .
	The following table lists the valid time interval strings and the meanings of each. The Format\$ function uses the same expressions.	
	Time	Interval
	"y"	Day of the year
	"yyyy"	Year
	"d"	Day
	"m"	Month
	"q"	Quarter
	"ww"	Week

	"h"	Hour
	"n"	Minute
	"s"	Second
	"w"	Weekday
	To find the number of days between two dates, you may use either day or day of the year, as they are both equivalent (" d ", " y ").	
	The time interval weekday (" w ") will return the number of weekdays occurring between date1 and date2, counting the first occurrence but not the last. However, if the time interval is week (" ww "), the function will return the number of calendar weeks between date1 and date2, counting the number of Sundays. If date1 falls on a Sunday, then that day is counted, but if date2 falls on a Sunday, it is not counted. The DateDiff function will return a negative date/time value if date1 is a date later in time than date2.	
Example	<p>This example gets today's date and adds ten days to it. It then calculates the difference between the two dates in days and weeks and displays the result.</p> <pre> Sub Main() Today\$ = Format(Date\$, "Short Date") NextWeek = Format(DateAdd("d", 14, today\$), "Short Date") DifDays# = DateDiff("d", today\$, NextWeek) DifWeek# = DateDiff("w", today\$, NextWeek) s\$ = "The difference between " & today\$ & " and " & NextWeek s\$ = s\$ & " is: " & DifDays# & " days or " & DifWeek# & " weeks" MsgBox s\$ End Sub </pre>	
See Also	DateAdd (on page 395) (function).	

DatePart (function)

Syntax	DatePart (interval\$,date)
Description	Returns an Integer representing a specific part of a date/time expression.

Com- ments	The <code>DatePart</code> function decomposes the specified date and returns a given date/time element. The following table describes the parameters:	
	Parameter	Description
	Interval\$	String expression indicating the specific time interval you wish to find the difference between.
	Date	Any expression convertible to a Date . An example of a valid date/time string would be " January 1, 1995 ".
	The following table lists the valid time interval strings and the meanings of each. The Format\$ function uses the same expressions.	
	Time	Interval
	"y"	Day of the year
	"yyyy"	Year
	"d"	Day
	"m"	Month
	"q"	Quarter
	"ww"	Week
	"h"	Hour
	"n"	Minute
	"s"	Second
	"w"	Weekday
	The weekday expression starts with Sunday as 1 and ends with Saturday as 7.	
Exam- ple	<p>This example displays the parts of the current date.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() today\$ = Date\$ qt = DatePart("q",today\$) yr = DatePart("yyyy",today\$) mo = DatePart("m",today\$) wk = DatePart("ww",today\$) </pre>	

	<pre> da = DatePart("d",today\$) s\$ = "The current date is:" & crlf & crlf s\$ = s\$ & "Quarter : " & qt & crlf s\$ = s\$ & "Year : " & yr & crlf s\$ = s\$ & "Month : " & mo & crlf s\$ = s\$ & "Week : " & wk & crlf s\$ = s\$ & "Day : " & da & crlf MsgBox s\$ End Sub </pre>
See Also	Day (on page 401) (function); Minute (on page 606) (function); Second (on page 705) (function); Month (on page 610) (function); Year (on page 797) (function); Hour (on page 549) (function); Weekday (on page 779) (function), Format (on page 525) (function).

DateSerial (function)

Syntax	<code>DateSerial</code> (year,month,day)	
Description	Returns a Date variant representing the specified date.	
Comments	The <code>DateSerial</code> function takes the following parameters:	
	Parameter	Description
	Year	Integer between 100 and 9999
	Month	Integer between 1 and 12
	Day	Integer between 1 and 31
Example	<p>This example converts a date to a real number representing the serial date in days since December 30, 1899 (which is day 0).</p> <pre> Sub Main() tdate# = DateSerial(1993,08,22) MsgBox "The DateSerial value for August 22, 1993, is: " & tdate# End Sub </pre>	
See Also	DateValue (on page 401) (function); TimeSerial (on page 757) (function); TimeValue (on page 758) (function); CDate , CVDate (on page 358) (functions).	

DateValue (function)

Syntax	DateValue (date_string\$)
Description	Returns a Date variant representing the date contained in the specified string argument.
Example	<p>This example returns the day of the month for today's date.</p> <pre> Sub Main() Tdate\$ = Date\$ tday\$ = DateValue(tdate\$) MsgBox "The date value of " & tdate\$ & " is: " & tday\$ End Sub </pre>
See Also	TimeSerial (on page 757) (function); TimeValue (on page 758) (function); DateSerial (on page 400) (function).
Platform(s)	All.

Day (function)

Syntax	Day (date)
Description	Returns the day of the month specified by date.
Comments	The value returned is an Integer between 0 and 31 inclusive. The date parameter is any expression that converts to a Date .
Example	<p>This example gets the current date and then displays it.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() CurDate = Now() MsgBox "Today is day " & Day(CurDate) & " of the month." & crlf & "Tomorrow is day " & Day(CurDate + 1) & "." End Sub </pre>
See Also	Minute (on page 606) (function); Second (on page 705) (function); Month (on page 610) (function); Year (on page 797) (function); Hour (on page 549) (function); Weekday (on page 779) (function); DatePart (on page 398) (function).

DDB (function)

Syntax	DDB (Cost, Salvage, Life, Period)	
Description	Calculates the depreciation of an asset for a specified Period of time using the double-declining balance method.	
Comments	<p>The double-declining balance method calculates the depreciation of an asset at an accelerated rate. The depreciation is at its highest in the first period and becomes progressively lower in each additional period. DDB uses the following formula to calculate the depreciation:</p> <pre style="background-color: #f0f0f0; padding: 5px;">DDB = ((Cost - Total_depreciation_from_all_other_periods) * 2) / Life</pre> <p>The DDB function uses the following parameters:</p>	
	Parameter	Description
	Cost	Double representing the initial cost of the asset.
	Salvage	Double representing the estimated value of the asset at the end of its predicted useful life
	Life	Double representing the predicted length of the asset's useful life
	Period	Double representing the period for which you wish to calculate the depreciation
	Life and Period must be expressed using the same units. For example, if Life is expressed in months, then Period must also be expressed in months.	
Example	<p>This example calculates the depreciation for capital equipment that cost \$10,000, has a service life of ten years, and is worth \$2,000 as scrap. The dialog box displays the depreciation for each of the first four years.</p> <pre style="background-color: #f0f0f0; padding: 5px;">Const crlf = Chr\$(13) + Chr\$(10) Sub Main() s\$ = "Depreciation Table" & crlf & crlf For yy = 1 To 4 CurDep# = DDB(10000.0,2000.0,10,yy) s\$ = s\$ & "Year " & yy & " : " & CurDep# & crlf Next yy</pre>	

	<pre>MsgBox s\$ End Sub</pre>
See Also	Sln (on page 720) (function); SYD (on page 745) (function).

DDEExecute (statement)

Syntax	DDEExecute channel, command\$						
Description	Executes a command in another application.						
Comments	The DDEExecute statement takes the following parameters:						
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Channel</td> <td>Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.</td> </tr> <tr> <td>Command\$</td> <td>String containing the command to be executed. The format of command\$ depends on the receiving application.</td> </tr> </tbody> </table>	Parameter	Description	Channel	Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.	Command\$	String containing the command to be executed. The format of command\$ depends on the receiving application.
Parameter	Description						
Channel	Integer containing the DDE channel number returned from DDEInitiate. An error will result if channel is invalid.						
Command\$	String containing the command to be executed. The format of command\$ depends on the receiving application.						
	If the receiving application does not execute the instructions, a runtime error is generated.						
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet. The command strings being created contain Microsoft Excel macro commands and may be concatenated and sent as one string to speed things up.</p> <pre>Sub Main() Dim cmd,q,ch% Q = Chr(34) ' Define quotation marks. Id = Shell("c:\excel5\excel.exe",3) 'Start Excel. ch% = DDEInitiate("Excel","Sheet1") On Error Resume Next cmd = "[ACTIVATE(" & q & "SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd</pre>						

	<pre> DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminate ch% Msgbox "Finished..." End Sub </pre>
See Also	DDEInitiate (on page 404) (function); DDEPoke (on page 405) (statement); DDERequest, DDERequest\$ (on page 407) (functions); DDESend (on page 408) (function); DDETerminate (on page 409) (statement); DDETerminateAll (on page 410) (statement); DDETimeout (on page 411) (statement).

DDEInitiate (function)

Syn-tax	DDEInitiate (application\$, topic\$)	
De-scrip-tion	Initializes a DDE link to another application and returns a unique number subsequently used to refer to the open DDE channel.	
Com-ments	The DDEInitiate statement takes the following parameters:	
	Para-meter	Description
	Ap-plica-tion\$	String containing the name of the application (the server) with which a DDE conversation will be established.
	Top-ic\$	String containing the name of the topic for the conversation. The possible values for this parameter are described in the documentation for the server application.
	This function returns 0 if the link cannot be established. This will occur under any of the following circumstances:	

	<ul style="list-style-type: none"> • The specified application is not running. • The topic was invalid for that application. • Memory or system resources are insufficient to establish the DDE link.
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet.</p> <pre> Sub Main() Dim cmd,q,ch% q = Chr(34) ' Define quotation marks. id = Shell("c:\excel5\excel.exe",3) 'Start Excel. ch% = DDEInitiate("Excel","Sheet1") On Error Resume Next cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminate ch% MsgBox "Finished..." End Sub </pre>
See Also	<p>DDEExecute (on page 403) (statement); DDEPoke (on page 405) (statement); DDERequest, DDERequest\$ (on page 407) (functions); DDESend (on page 408) (function); DDETerminate (on page 409) (statement); DDETerminateAll (on page 410) (statement); DDETimeout (on page 411) (statement).</p>

DDEPoke (statement)

Syntax	DDEPoke channel, Dataltem, value
Description	Sets the value of a data item in the receiving application associated with an open DDE link.

Comments	The DDEPoke statement takes the following parameters:	
	Parameter	Description
	Channel	Integer containing the DDE channel number returned from <code>DDEInitiate</code> . An error will result if channel is invalid.
	DataItem	Data item to be set. This parameter can be any expression convertible to a String . The format depends on the server.
	Value	The new value for the data item. This parameter can be any expression convertible to a String . The format depends on the server. A runtime error is generated if value is Null.
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet.</p> <pre> Sub Main() Dim cmd,q,ch% Q = Chr(34) ' Define quotation marks. Id = Shell("c:\excel5\excel.exe",3) 'Start Excel. Ch% = DDEInitiate("Excel","Sheet1") On Error Resume Next cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminate ch% MsgBox "Finished..." End Sub </pre>	
See Also	DDEExecute (on page 403) (statement); DDEInitiate (on page 404) (function); DDERequest , DDERequest\$ (on page 407) (functions); DDESend (on page 408) (function); DDETerminate (on page 409) (statement); DDETerminateAll (on page 410) (statement); DDETimeout (on page 411) (statement).	

DDERequest, DDERequest\$ (functions)

Syntax	DDERequest [\$](channel,DataItem\$)	
Description	Returns the value of the given data item in the receiving application associated with the open DDE channel.	
Comments	DDERequest\$ returns a String , whereas DDERequest returns a String variant. The DDERequest/DDERequest\$ functions take the following parameters:	
	Parameter	Description
	channel	Integer containing the DDE channel number returned from <code>DDEInitiate</code> . An error will result if channel is invalid.
	DataItem\$	String containing the name of the data item to request. The format for this parameter depends on the server.
	The format for the returned value depends on the server.	
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet.</p> <pre> Sub Main() Dim cmd,q,ch% q = Chr(34) ' Define quotation marks. id = Shell("c:\excel5\excel.exe",3) 'Start Excel. ch% = DDEInitiate("Excel","Sheet1") On Error Resume Next cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminate ch% </pre>	

	<pre>Msgbox "Finished..." End Sub</pre>
See Also	DDEExecute (on page 403) (statement); DDEInitiate (on page 404) (function); DDEPoke (on page 405) (statement); DDESend (on page 408) (function); DDETerminate (on page 409) (statement); DDETerminateAll (on page 410) (statement); DDETimeout (on page 411) (statement).

DDESend (statement)

Syntax	DDESend application\$, topic\$, Dataltem, value	
Description	Initiates a DDE conversation with the server as specified by application\$ and topic\$ and sends that server a new value for the specified item.	
Comments	The DDESend statement takes the following parameters:	
	Parameter	Description
	application\$	String containing the name of the application (the server) with which a DDE conversation will be established.
	topic\$	String containing the name of the topic for the conversation. The possible values for this parameter are described in the documentation for the server application.
	Dataltem	Data item to be set. This parameter can be any expression convertible to a String . The format depends on the server.
	value	New value for the data item. This parameter can be any expression convertible to a String . The format depends on the server. A runtime error is generated if value is Null.
	<p>The DDESend statement performs the equivalent of the following statements:</p> <pre>ch% = DDEInitiate(application\$,topic\$) DDEPoke ch%,item,data DDETerminate ch%</pre>	
Example	This example sets the content of the first cell in an Excel spreadsheet.	

	<pre> Sub Main() Dim cmd,ch% id = Shell("c:\excel15\excel.exe",3) 'Start Excel. On Error Goto ExcelError DDESend "Excel","Sheet1","R1C1","Payroll For August 1995" MsgBox "Finished " Exit Sub ExcelError: MsgBox "Error sending data to Excel." Exit Sub 'Reset error handler. End Sub </pre>
See Also	<p>DDEExecute (on page 403) (statement); DDEInitiate (on page 404) (function); DDEPoke (on page 405) (statement); DDERequest (on page 407), DDERequest\$ (on page 407) (functions); DDETerminate (on page 409) (statement); DDETerminateAll (on page 410) (statement); DDETimeout (on page 411) (statement).</p>

DDETerminate (statement)

Syntax	DDETerminate channel
Description	Closes the specified DDE channel.
Comments	The channel parameter is an Integer containing the DDE channel number returned from DDEInitiate . An error will result if channel is invalid. All open DDE channels are automatically terminated when the script ends.
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet.</p> <pre> Sub Main() Dim cmd,q,ch% q = Chr(34) ' Define quotation marks. Id = Shell("c:\excel15\excel.exe",3) 'Start Excel. Ch% = DDEInitiate("Excel","Sheet1") </pre>

	<pre> On Error Resume Next cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminate ch% Msgbox "Finished..." End Sub </pre>
See Also	<p>DDEExecute (on page 403) (statement); DDEInitiate (on page 404) (function); DDEPoke (on page 405) (statement); DDERequest (on page 407), DDERequest\$ (on page 407) (functions); DDESend (on page 408) (function); DDETerminateAll (on page 410) (statement); DDETimeout (on page 411) (statement).</p>

DDETerminateAll (statement)

Syntax	DDETerminateAll
Description	Closes all open DDE channels.
Comments	All open DDE channels are automatically terminated when the script ends.
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet.</p> <pre> Sub Main() Dim cmd,q,ch% q = Chr(34) ' Define quotation marks. id = Shell("c:\excel5\excel.exe",3) 'Start Excel. ch% = DDEInitiate("Excel","Sheet1") On Error Resume Next </pre>

	<pre> cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminateAll Msgbox "Finished " End Sub </pre>
See Also	DDEExecute (on page 403) (statement); DDEInitiate (on page 404) (function); DDEPoke (on page 405) (statement); DDERequest (on page 407) , DDERequest\$ (on page 407) (functions); DDESend (on page 408) (function); DDETerminate (on page 409) (statement); DDETimeout (on page 411) (statement).


DDETimeout (statement)


Syntax	DDETimeout milliseconds
Description	Sets the number of milliseconds that must elapse before any DDE command times out.
Comments	The milliseconds parameter is a Long and must be within the following range: 0 <= milliseconds <= 2,147,483,647 The default is 10,000 (10 seconds).
Example	<p>This example sets and retrieves a cell in an Excel spreadsheet. The timeout has been set to wait 2 seconds for Excel to respond before timing out.</p> <pre> Sub Main() Dim cmd,q,ch% q = Chr(34) ' Define quotation marks. id = Shell("c:\excel5\excel.exe",3) 'Start Excel. ch% = DDEInitiate("Excel","Sheet1") DDETimeout 2000 'Wait 2 seconds for Excel to respond On Error Resume Next </pre>

	<pre>cmd = "[ACTIVATE(" & q &"SHEET1" & q & ")]" 'Activate worksheet. DDEExecute ch%,cmd DDEPoke ch%,"R1C1","\$1000.00" 'Send value to cell. 'Retrieve value and display. MsgBox "The value of Row 1, Cell 1 is: " & DDERequest(ch%,"R1C1") DDETerminate ch% Msgbox "Finished " End Sub</pre>
See Also	DDEExecute (on page 403) (statement); DDEInitiate (on page 404) (function); DDEPoke (on page 405) (statement); DDERequest (on page 407) , DDERequest\$ (on page 407) (functions); DDESend (on page 408) (function); DDETerminate (on page 409) (statement); DDE-TerminateAll (on page 410) (statement).

Declare (statement)

Syntax	Declare {Sub Function} name[TypeChar] [CDecl Pascal System StdCal [Lib "LibName\$" [Alias "AliasName\$"]] [(ParameterList))] [As type] Where ParameterList is a comma-separated list of the following (up to 30 [Optional] [ByVal ByRef] ParameterName[()] [As ParameterType]	
Description	Creates a prototype for either an external routine or a Basic Control Engine in the source module or in another source module.	
Comments	<p>Declare statements must appear outside of any Sub or Function declaration.</p> <p>Declare statements are only valid during the life of the script in which they</p> <p>The Declare statement uses the following parameters:</p>	
	Parameter	Description
	name	Any valid script name. When you include a type-declaration character type.

		This name is specified as a normal identifier and does not appear within quotes.
	TypeChar	<p>An optional type-declaration character that specifies the type of data returned from functions. The following characters: #, !, \$, @ are allowed. In most situations, the @ character is not allowed.</p> <p>Type-declaration characters can be used in variable declarations, and take the place of the type name.</p> <div style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; margin-top: 10px;">  Note: Currency data cannot be returned from most functions. Thus, the @ type-declaration character cannot be used when declaring a currency variable. </div>
	CDecl	Optional keyword indicating that the routine or function uses the C calling convention. Arguments are pushed right to left on the stack, and the called function performs stack cleanup.
	Pascal	Optional keyword indicating that the routine or function uses the Pascal calling convention. In Pascal routines, arguments are pushed left to right, and the called function performs stack cleanup.
	System	Optional keyword indicating that the routine or function uses the System calling convention. In System routines, arguments are pushed right to left, and the caller performs stack cleanup. The register used for arguments is specified in the AL register.
	StdCall	Optional keyword indicating that the routine or function uses the StdCall calling convention. In StdCall routines, arguments are pushed right to left, and the called function performs stack cleanup.
	LibName\$	Must be specified if the routine is external. It specifies the name of the library containing the external routine and must be followed by a dollar sign (\$).

		The LibName\$ parameter can include a path, specifying the exact location of the library.
	AliasName\$	Alias name that must be given to the routine if the name parameter is not used. For example, the following two statements declare the same routine:
		<pre> Declare Function GetCurrentTime Lib "user" Alias "GetCurrentTime" _ As Integer Declare Function GetTime Lib "user" Alias "GetCurrentTime" _ As Integer </pre>
		Use an alias when the name of an external routine with the name of an internal routine. If the routine name contains invalid characters, use an alias. The AliasName\$ parameter must be the same as the name of the routine.
	type	Indicates the return type for functions. For external functions, the valid return types are Long, String, Single, Double, Date, and Object . Objects are not supported. <div style="border: 1px solid #0070c0; border-radius: 10px; padding: 10px; margin-top: 10px;">  Note: Currency, Variant, fixed-length strings, user-defined types, and OLE automation objects cannot be returned by external functions. </div>
	Optional	Keyword indicating that the parameters that follow the first optional parameter must be of type optional. If this keyword is omitted, then the parameters are required when calling this subroutine.

	ByVal	Optional keyword indicating that the parameter is passed by value. Parameters passed by value cannot be changed by the called routine.
	ByRef	Optional keyword indicating that the parameter is passed by reference. Parameters passed by reference can be changed by the called routine. If no keywords are specified, then ByRef is assumed.
	ParameterName	Name of the parameter, which must follow the following conventions: <ol style="list-style-type: none"> 1. Must start with a letter. 2. May contain letters, digits, and underscores (_). Punctuation and type-declaration characters are not allowed. The exclamation point (!) is allowed as the last character in the name as long as it is not the last character in a type declaration. The exclamation point is interpreted as a type-declaration character. 3. Must not exceed 80 characters. Additionally, ParameterName can contain a type-declaration character specifying the parameter's type (that is, any of the following characters):
	()	Indicates that the parameter is an array.
	ParameterType	Specifies the type of the parameter (such as String , Variant , and so on). The Any keyword should only be included if ParameterType is a type-declaration character. <p>In addition to the default data types, you can specify any user-defined structure or automation object. If the data type cannot be known in advance, then the Any keyword forces the compiler to relax type checking and allow a type to be passed in place of the specified type.</p>
		Declare Sub Convert Lib "mylib" (a

The **Any** data type can only be used for pointers to external routines.

Passing Parameters

By default, arguments are passed by reference. Many external routines require a pointer to a value. The `ByVal` keyword does this. For example, this C routine

```
void MessageBeep(int);
```

would be declared as follows:

```
Declare Sub MessageBeep Lib "user" (ByVal n As Integer)
```

As an example of passing parameters by reference, consider the following routine that takes a pointer to an integer as the third parameter:

```
int SystemParametersInfo(int,int,int *,int);
```

This routine would be declared as follows (notice the `ByRef` keyword in the declaration):

```
Declare Function SystemParametersInfo Lib "user" (ByVal action As Integer, _
    ByVal uParam As Integer,ByRef pInfo As Integer, _
    ByVal updateINI As Integer) As Integer
```

Strings can be passed by reference or by value. When they are passed by reference, an internal handle to the string is passed. When they are passed by value, the routine receives a pointer to a null-terminated string (that is, a C string). If an external routine requires a string variable, then there must be sufficient space within the string to hold the result. This can be accomplished using the `Space` function, as shown in the following example:

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal dirname$,ByVal length%)
:
Dim s As String
s = Space(128)
GetWindowsDirectory s,128
```

Another alternative to ensure that a string has sufficient space is to declare the string with a fixed length:

```
Declare Sub GetWindowsDirectory Lib "kernel" (ByVal dirname$,ByVal length%)
:
```

```
Dim s As String * 128 'Declare a fixed-length string.
GetWindowsDirectory s,len(s) 'Pass it to an external subroutine.
```

Calling Conventions with External Routines

For external routines, the argument list must exactly match that of the routine being called. When calling an external subroutine or function, the script needs to be told how that routine passes its parameters and who is responsible for cleanup of the stack.

The following table describes which calling conventions are supported on the platform. It also indicates what the default calling convention is when no explicit calling convention is declared in the declare statement.

Passing Null Pointers

To pass a null pointer to an external procedure, declare the parameter that receives the pointer as type Any, then pass a long value 0 by value:

```
Declare Sub Foo Lib "sample" (ByVal lpName As Any)
Sub Main()
Sub Foo "Hello" 'Pass a 32-bit pointer to a null-terminated string
Sub Foo ByVal 0& 'Pass a null pointer
End Sub
```

Passing Data to External Routines

The following table shows how the different data types are passed to external routines.

Data Type	Is Passed As
ByRef Boolean	A 32-bit pointer to a 2-byte value.
ByVal Boolean	A 2-byte value containing -1 or 0.
ByVal Integer	A 32-bit pointer to a 2-byte short integer.
ByRef Integer	A 2-byte short integer.
ByVal Long	A 32-bit pointer to a 4-byte long integer.
ByRef Long	A 4-byte long integer.
ByRef Single	A 32-bit pointer to a 4-byte IEEE floating-point value.
ByVal Single	A 4-byte IEEE floating-point value.

	ByRef Double	A 32-bit pointer to an 8-byte IEEE floating-point value (IEEE 754 double).
	ByVal Double	An 8-byte IEEE floating-point value (IEEE 754 double).
	ByVal String	A 32-bit pointer to a null-terminated string containing embedded nulls (Chr\$(0)). The first null represents the end of the string. The first null is considered the string terminator. An external routine can freely change the string. It cannot, however, write beyond the terminator.
	ByRef String	A 32-bit pointer to a 2-byte internal string. This value can only be used for strings written specifically for the Basic Control Engine.
	ByRef Date	A 32-bit pointer to an 8-byte IEEE floating-point value (IEEE 754 double).
	ByVal Date	An 8-byte IEEE floating-point value (IEEE 754 double).
	ByRef Currency	A 32-bit pointer to an 8-byte integer scaled by 10000.
	ByVal Currency	An 8-byte integer scaled by 10000.
	ByRef Variant	A 32-bit pointer to a 16-byte internal variant structure. The structure contains a 2-byte type (returned by the VarType function), followed by 14 bytes of data (aligned to 8 bytes).
	ByVal Variant	A 16-byte variant structure. This structure contains a 2-byte type (the same as that returned by VarType), followed by 6 bytes of slop (for alignment), followed by 8 bytes containing the value.
	ByVal Object	For data objects, a 32-bit pointer to an integer. This value can only be used for objects written specifically for the Basic Control Engine. For OLE automation objects, a 32-bit PATCH handle is passed.

	<p>ByRef Object</p>	<p>For data objects, a 32-bit pointer to the object is passed as an integer that references the object. This value can be used by external routines written for the Basic Control Engine.</p> <p>For OLE automation objects, a 32-bit pointer to the object's internal ID is passed. This value can be used by external routines written specifically for the Basic Control Engine.</p>
	<p>User-defined type</p>	<p>A 32-bit pointer to the structure. User-defined structures can be passed by reference.</p> <p>It is important to remember that structures in the Basic Control Engine scripts are packed on 2-bytes. This means that the individual structure members are aligned consistently with similar structures in other languages.</p>
	<p>Arrays</p>	<p>A 32-bit pointer to a packed array of the specified type. Arrays can only be passed by reference.</p>
	<p>Dialogs</p>	<p>Dialogs cannot be passed to external routines.</p>
	<p>Only variable-length strings can be passed to external routines; fixed-length strings are converted to variable-length strings.</p>	
	<p>The Basic Control Engine passes data to external functions consistent with the language as defined by the Declare statement. There is one exception to this rule: you can pass individual parameters using the ByVal keyword when passing individual parameters. The following example shows a number of different ways to pass an Integer to an external routine:</p>	
	<pre> Declare Sub Foo Lib "MyLib" (ByRef i As Integer) Sub Main Dim i As Integer i = 6 Foo 6 'Passes a temporary integer (value 6) by reference Foo i 'Passes variable "i" by reference Foo (i) 'Passes a temporary integer (value 6) by reference Foo i + 1 'Passes temporary integer (value 7) by reference Foo ByVal i 'Passes i by value End Sub </pre>	

The above example shows that the only way to override passing a value by Val keyword.

Note: Use caution when using the ByVal keyword in this way. The external receive a pointer to an Integer—a 32-bit value; using ByVal causes the Basic Integer by value—a 16-bit value. Passing data of the wrong size to any external results in unpredictable results.

Example

```

Declare Function GetProfileString Lib "kernel32" Alias
"GetProfileStringA" (ByVal lpAppName As String, ByVal lpKeyName As
String, ByVal lpDefault As String, ByVal lpReturnedString As
String, ByVal nSize As Long) As Long
Sub Main()
SName$ = "Intl" 'Win.ini section name.
KName$ = "sCountry" 'Win.ini country setting.
ret$ = String(255,0) 'Initialize return string.
myvalue = GetProfileString( SName$, KName$, "", ret$, Len(ret$) )
If myvalue Then
MsgBox "Your country setting is: " & ret$
Else
MsgBox "There is no country setting in your win.ini file."
End If
End Sub
    
```

See Also

[Call \(statement\) \(on page 355\)](#), [Sub...End Sub \(statement\) \(on page 744\)](#), [Function \(statement\) \(on page 533\)](#).

Notes

Under Win32, external routines are contained in DLLs. The libraries contained in the script are loaded when the routine is called for the first time (that is, not when the script is loaded). This means that you must reference external DLLs that potentially do not exist.

All the Win32 API routines are contained in DLLs, such as "user32", "kernel32", etc. The extension ".exe" is implied if another extension is not given.

The **Pascal** and **StdCall** calling conventions are identical on Win32 platform. On the Win64 platform, the arguments are passed using C ordering regardless of the calling convention. The arguments are left on the stack.

If the **libname\$** parameter does not contain an explicit path to the DLL, the following search is performed for the DLL (in this order):

1. The directory containing the Basic Control Engine scripts.
2. The current directory.
3. The Windows system directory.
4. The Windows directory.
5. All directories listed in the path environment variable.

If the first character of aliasname\$ is #, then the remainder of the character is the number of the routine to be called. For example, the following two statements define Win32, GetCurrentTime is defined as GetTickCount, ordinal 300, in kernel32.dll.

```
Declare Function GetTime Lib "kernel32.dll" Alias "GetTickCount" () As Long
Declare Function GetTime Lib "kernel32.dll" Alias "#300" () As Long
```

DefType (statement)

Syntax	DefInt letterrange DefLng letterrange DefStr letterrange DefSng letterrange DefDbl letterrange DefCur letterrange DefObj letterrange DefVar letterrange DefBool letterrange DefDate letterrange
Description	Establishes the default type assigned to undeclared or untyped variables.
Comments	The Def Type statement controls automatic type declaration of variables. Normally, if a variable is encountered that hasn't yet been declared with the Dim , Public , or Private statement or does not appear with an explicit type-declaration character, then that variable is declared implicitly as a variant (DefVar A-Z). This can be changed using the Def Type statement to specify starting letter ranges for type other than integer. The letterrange parameter is used to specify starting letters. Thus, any variable that begins with a specified character will be declared using the specified Type.
	The syntax for letterrange is: letter [-letter] [,letter [-letter]]... Def Type variable types are superseded by an explicit type declaration ³⁴ using either a type-declaration character or the Dim , Public , or Private statement.

	<p>The Def Type statement only affects how the Basic Control Engine compiles scripts and has no effect at runtime. The Def Type statement can only appear outside all Sub and Function declarations. The following table describes the data types referenced by the different variations of the Def Type statement:</p>	
	Statement	Data Type
	DefInt	Integer
	DefLng	Long
	DefStr	String
	DefSng	Single
	DefDbl	Double
	DefCur	Currency
	DefObj	Object
	DefVar	Variant
	DefBool	Boolean
	DefDate	Date
Example	<pre> DefStr a-m DefLng n-r DefSng s-u DefDbl v-w DefInt x-z </pre>	
	<pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a = 100.52 n = 100.52 s = 100.52 v = 100.52 x = 100.52 msg1 = "The values are:" & crlf & crlf msg1 = msg1 & "(String) a: " & a & crlf msg1 = msg1 & "(Long) n: " & n & crlf msg1 = msg1 & "(Single) s: " & s & crlf msg1 = msg1 & "(Double) v: " & v & crlf </pre>	

	<pre>msg1 = msg1 & "(Integer) x: " & x & crlf MsgBox msg1 End Sub</pre>
See Also	Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Long (on page 598) (data type); Object (on page 633) (data type); Single (on page 718) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); Integer (on page 566) (data type).

DeleteSetting (statement)

Syntax	DeleteSetting appname [,section [,key]]	
Description	Deletes a setting from the registry.	
Comments	You can control the behavior of <code>DeleteSetting</code> by omitting parameters. If you specify all three parameters, then <code>DeleteSetting</code> deletes your specified setting. If you omit key, then <code>DeleteSetting</code> deletes all of the keys from section. If both section and key are omitted, then <code>DeleteSetting</code> removes that application's entry from the system registry. The following table describes the named parameters to the <code>DeleteSetting</code> statement:	
	Parameter	Description
	appname	String expression indicating the name of the application whose setting will be deleted.
	section	String expression indicating the name of the section whose setting will be deleted.
	key	String expression indicating the name of the setting to be deleted from the registry.
Example	<pre>'The following example adds two entries to the Windows registry 'if run under Win32 or to NEWAPP.INI on other platforms, 'using the SaveSetting statement. It then uses DeleteSetting 'first to remove the Startup section, then to remove 'the NewApp key altogether. Sub Main() SaveSetting appname := "NewApp", section := "Startup", _</pre>	

	<pre> key := "Height", setting := 200 SaveSetting appname := "NewApp", section := "Startup", _ key := "Width", setting := 320 DeleteSetting "NewApp", "Startup" 'Remove Startup section DeleteSetting "NewApp" 'Remove NewApp key End Sub </pre>
See Also	SaveSetting (on page 701) (statement), GetSetting (on page 541) (function), GetAllSettings (on page 537) (function)
Notes	Under Win32, this statement operates on the system registry. All settings are saved under the following entry in the system registry: HKEY_CURRENT_USER\Software\BasicScript Program Settings\appname\section\key

Dialog (function)

Syn-tax	Dialog (DialogVariable [,DefaultButton] [,Timeout])				
De-scrip-tion	Displays the dialog box associated with DialogVariable, returning an Integer indicating which button was clicked.				
Com-ments	The function returns any of the following values:				
	-1 The OK button was clicked.				
	0 The Cancel button was clicked.				
	>0 A push button was clicked. The returned number represents which button was clicked based on its order in the dialog box template (1 is the first push button, 2 is the second push button, and so on).				
	The Dialog function accepts the following parameters:				
	<table border="1"> <thead> <tr> <th>Parame-ter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Dialog-Variable</td> <td>Name of a variable that has previously been dimensioned as a user dialog box. This is accomplished using the Dim statement:</td> </tr> </tbody> </table>	Parame-ter	Description	Dialog-Variable	Name of a variable that has previously been dimensioned as a user dialog box. This is accomplished using the Dim statement:
Parame-ter	Description				
Dialog-Variable	Name of a variable that has previously been dimensioned as a user dialog box. This is accomplished using the Dim statement:				

		<pre>Dim MyDialog As MyTemplate</pre> <p>All dialog variables are local to the Sub or Function in which they are defined. Private and public dialog variables are not allowed.</p>
	Default-Button	An Integer specifying which button is to act as the default button in the dialog box. The value of DefaultButton can be any of the following:
		-2 This value indicates that there is no default button.
		-1 This value indicates that the OK button, if present, should be used as the default.
		0 This value indicates that the Cancel button, if present, should be used as the default.
		>0 This value indicates that the Nth button should be used as the default. This number is the index of a push button within the dialog box template.
		If DefaultButton is not specified, then -1 is used. If the number specified by DefaultButton does not correspond to an existing button, then there will be no default button. The default button appears with a thick border and is selected when the user presses Enter on a control other than a push button.
	Timeout	An Integer specifying the number of milliseconds to display the dialog box before automatically dismissing it. If Timeout is not specified or is equal to 0 , then the dialog box will be displayed until dismissed by the user. If a dialog box has been dismissed due to a timeout, the Dialog function returns 0 .
Example	<p>This example displays an abort/retry/ignore disk error dialog box.</p> <pre>Sub Main() Begin Dialog DiskErrorTemplate 16,32,152,48,"Disk Error" Text 8,8,100,8,"The disk drive door is open." PushButton 8,24,40,14,"Abort",.Abort PushButton 56,24,40,14,"Retry",.Retry PushButton 104,24,40,14,"Ignore",.Ignore End Dialog Dim DiskError As DiskErrorTemplate r% = Dialog(DiskError,3,0) MsgBox "You selected button: " & r% End Sub</pre>	
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 426) (statement); DlgProc (on page 440) (func-	

tion); [DropListBox \(on page 458\)](#) (statement); [GroupBox \(on page 544\)](#) (statement); [ListBox \(on page 591\)](#) (statement); [OKButton \(on page 638\)](#) (statement); [OptionButton \(on page 652\)](#) (statement); [OptionGroup \(on page 653\)](#) (statement); [Picture \(on page 657\)](#) (statement); [PushButton \(on page 671\)](#) (statement); [Text \(on page 752\)](#) (statement); [TextBox \(on page 753\)](#) (statement); [Begin \(on page 348\)](#) [Dialog \(on page 348\)](#) (statement), [PictureButton \(on page 659\)](#) (statement).

Dialog (statement)

Syntax	Dialog DialogVariable [, [DefaultButton] [, Timeout]]
Description	Same as the Dialog (on page 424) function, except that the Dialog statement does not return a value.
Example	<p>This example displays an Abort/Retry/Ignore disk error dialog box.</p> <pre> Sub Main() Begin Dialog DiskErrorTemplate 16,32,152,48,"Disk Error" Text 8,8,100,8,"The disk drive door is open." PushButton 8,24,40,14,"Abort",.Abort PushButton 56,24,40,14,"Retry",.Retry PushButton 104,24,40,14,"Ignore",.Ignore End Dialog Dim DiskError As DiskErrorTemplate Dialog DiskError,3,0 End Sub </pre>
See Also	Dialog (on page 424) (function); DlgProc (on page 440) (function)

Dim (statement)

Naming Conventions Variable names must follow these naming rules:

1. Must start with a letter.
2. May contain letters, digits, and the underscore character (_); punctuation is not allowed. The exclamation point (!) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character.
3. The last character of the name can be any of the following type-declaration characters: # , @ , % , ! , & , and \$.

4. Must not exceed 80 characters in length.
5. Cannot be a reserved word. For example, **Project**.

Dir, Dir\$ (functions)

Syntax	Dir\$ [(filespec\$ [attributes])]		
Description	Returns a String containing the first or next file matching filespec\$. If filespec\$ is specified, then the first file matching that filespec\$ is returned. If filespec\$ is not specified, then the next file matching the initial filespec\$ is returned.		
Comments	Dir\$ returns a String , whereas Dir returns a String variant. The Dir\$ / Dir functions take the following parameters:		
	Parameter	Description	
	filespec\$	String containing a file specification. If this parameter is specified, then Dir\$ returns the first file matching this file specification. If this parameter is omitted, then the next file matching the initial file specification is returned. If no path is specified in filespec\$, then the current directory is used.	
	attributes	Integer specifying attributes of files you want included in the list, as described below. If omitted, then only the normal, read-only, and archive files are returned.	
	An error is generated if Dir\$ is called without first calling it with a valid filespec\$. If there is no matching filespec\$, then a zero-length string is returned.		
	Wildcards The filespec\$ argument can include wildcards, such as * and ?. The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complete searching patterns. The following table shows some examples:		
	This pattern	Matches these files	Doesn't match these files
	S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
	C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
	C*T	CAT CAP.TXT	CAT.DOC
	C?T	CAT CUT	CAT.TXT CAPIT CT
	*	(All files)	

	Attributes You can control which files are included in the search by specifying the optional attributes parameter. The Dir , Dir\$ functions always return all normal, read-only, and archive files (ebNormal Or ebReadOnly Or ebArchive). To include additional files, you can specify any combination of the following attributes (combined with the Or operator):		
	Constant	Value	Includes
	ebNormal	0	Normal, Read-only, and archive files
	ebHidden	2	Hidden files
	ebSystem	4	System files
	ebVolume	8	Volume label
	ebDirectory	16	DOS subdirectories
Example	<p>This example uses Dir to fill a SelectBox with the first 10 directory entries.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Option Base 1 Sub Main() Dim a\$(10) i% = 1 a(i%) = Dir("*.**") While (a(i%) <> "") and (i% < 10) i% = i% + 1 a(i%) = Dir Wend r = SelectBox("Top 10 Directory Entries",,a) End Sub </pre>		
See Also	ChDir (on page 359) (statement); ChDrive (on page 359) (statement); CurDir, CurDir\$ (on page 387) (functions); MkDir (on page 608) (statement); Rmdir (on page 694) (statement); FileList (on page 517) (statement).		

DiskDrives (statement)

Syntax	DiskDrives array()
--------	---------------------------

De- scrip- tion	Fills the specified String or Variant array with a list of valid drive letters.
Com- ments	The array () parameter specifies either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed. If array () is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound , UBound , and Array-Dims functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.
Exam- ple	This example builds and displays an array containing the first three available disk drives. <pre>Sub Main() Dim drive\$() DiskDrives drive\$ r% = SelectBox("Available Disk Drives", ,drive\$) End Sub</pre>
See Also	ChDrive (on page 359) (statement); DiskFree (on page 429) (function).

DiskFree (function)

Syntax	DiskFree& ([drive\$])
Descrip- tion	Returns a Long containing the free space (in bytes) available on the specified drive.
Com- ments	If drive\$ is zero-length or not specified, then the current drive is assumed. Only the first character of the drive\$ string is used.
Exam- ple	This example uses DiskFree to set the value of i and then displays the result in a message box. <pre>Sub Main() s\$ = "c" i# = DiskFree(s\$)</pre>

	<pre>MsgBox "Free disk space on drive '" & s\$ & "' is: " & i# End Sub</pre>
See Also	ChDrive (on page 359) (statement); DiskDrives (on page 428) (statement).

DlgCaption (function)

Syntax	<code>DlgCaption[()]</code>
Description	Returns a string containing the caption of the active user-defined dialog box.
Comments	This function returns a zero-length string if the active dialog has no caption.
See Also	Begin Dialog (on page 348) (statement)

DlgCaption (statement)

Syntax	<code>DlgCaption text</code>
Description	Changes the caption of the current dialog to text.
Example	<pre> 'This example displays a dialog box, adjusting the caption 'to contain the text of the currently selected option 'button. Function DlgProc(c As String,a As Integer,v As Integer) If a = 1 Then DlgCaption choose(DlgValue("OptionGroup1") + 1, _ "Blue","Green") ElseIf a = 2 Then DlgCaption choose(DlgValue("OptionGroup1") + 1, _ "Blue","Green") End If End Function Sub Main() Begin Dialog UserDialog ,,149,45,"Untitled",.DlgProc OKButton 96,8,40,14 OptionGroup .OptionGroup1 </pre>

	<pre> OptionButton 12,12,56,8,"Blue",.OptionButton1 OptionButton 12,28,56,8,"Green",.OptionButton2 End Dialog Dim d As UserDialog Dialog d End Sub </pre>
See Also	Begin Dialog (on page 348) (statement)

DlgControlId (function)

Syn- tax	DlgControlId (ControlName\$)
De- scrip- tion	Returns an Integer containing the index of the specified control as it appears in the dialog box template.
Com- ments	The first control in the dialog box template is at index 0, the second is at index 1, and so on. The ControlName\$ parameter contains the name of the .Identifier parameter associated with that control in the dialog box template.
	The Basic Control Engine statements and functions that dynamically manipulate dialog box controls identify individual controls using either the .Identifier name of the control or the control's index. Using the index to refer to a control is slightly faster but results in code that is more difficult to maintain.
Exam- ple	<p>This example uses DlgControlId to verify which control was triggered and branches the dynamic dialog script accordingly.</p> <pre> Function DlgProc(ControlName\$,Action%,SuppValue%) As Integer If Action% = 2 Then 'Enable the next three controls. If DlgControlId(ControlName\$) = 2 Then For i = 3 to 5 DlgEnable i,DlgValue("CheckBox1") Next i DlgProc = 1 'Don't close the dialog box. End If ElseIf Action% = 1 Then </pre>

	<pre>'Set initial state upon startup For i = 3 to 5 DlgEnable i,DlgValue("CheckBox1") Next i End If End Function</pre>
	<pre>Sub Main() Begin Dialog UserDialog , ,180,96,"Untitled",.DlgProc OKButton 132,8,40,14 CancelButton 132,28,40,14 CheckBox 24,16,72,8,"Click Here",.CheckBox1 CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2 CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3 CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4 CheckBox 24,72,76,8,"Main Option 2",.CheckBox5 End Dialog Dim d As UserDialog Dialog d End Sub</pre>
See Also	<p>DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).</p>

DlgEnable (function)

Syn-tax	DlgEnable (ControlName\$ ControlIndex)
De-scrip-tion	Returns True if the specified control is enabled; returns False otherwise.
Com-ments	Disabled controls are dimmed and cannot receive keyboard or mouse input. The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dia-

log box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the `ControllIndex` parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on). You cannot disable the control with the focus.

**Exam-
ple**

This example checks the status of a checkbox at the end of the dialog procedure and notifies the user accordingly.

```
Function DlgProc(ControlName$,Action%,SuppValue%) As Integer

    If Action% = 2 Then

        'Enable the next three controls.

        If DlgControlId(ControlName$) = 2 Then

            For i = 3 to 5

                DlgEnable i,DlgValue("CheckBox1")

            Next i

            DlgProc = 1 'Don't close the dialog box.

        End If

    ElseIf Action% = 1 Then

        'Set initial state upon startup

        For i = 3 to 5

            DlgEnable i,DlgValue("CheckBox1")

        Next i

    End If

    If DlgEnable(i) = True Then

        MsgBox "You do not have the required disk space.",vbExclamation,"Insufficient Disk Space"

    End If

End Function
```

```
Sub Main()

    Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc

        OKButton 132,8,40,14

        CancelButton 132,28,40,14

        CheckBox 24,16,72,8,"Click Here",.CheckBox1

        CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2

        CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3

        CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4

        CheckBox 24,72,76,8,"Main Option 2",.CheckBox5

    End Dialog
```

	<pre>Dim d As UserDialog Dialog d End Sub</pre>
See Also	<p>DlgControl (on page 431) (statement); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).</p>

DlgEnable (statement)

Syntax	DlgEnable {ControlName\$ ControlIndex} [,isOn]
Description	Enables or disables the specified control.
Comments	Disabled controls are dimmed and cannot receive keyboard or mouse input. The isOn parameter is an Integer specifying the new state of the control. It can be any of the following values: 0 The control is disabled. 1 The control is enabled. Omitted Toggles the control between enabled and disabled.
	Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).
	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
Example	<p>This example uses DlgEnable to turn on/off various dialog options.</p> <pre>Function DlgProc(ControlName\$,Action%,SuppValue%) As Integer If Action% = 2 Then 'Enable the next three controls. If DlgControlId(ControlName\$) = 2 Then For i = 3 to 5 DlgEnable i,DlgValue("CheckBox1") Next i End If End If End Function</pre>

	<pre> Next i DlgProc = 1 'Don't close the dialog box. End If ElseIf Action% = 1 Then 'Set initial state upon startup For i = 3 to 5 DlgEnable i,DlgValue("CheckBox1") Next i End If End Function </pre>
	<pre> Sub Main() Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc OKButton 132,8,40,14 CancelButton 132,28,40,14 CheckBox 24,16,72,8,"Click Here",.CheckBox1 CheckBox 36,32,60,8,"Sub Option 1",.CheckBox2 CheckBox 36,44,72,8,"Sub Option 2",.CheckBox3 CheckBox 36,56,60,8,"Sub Option 3",.CheckBox4 CheckBox 24,72,76,8,"Main Option 2",.CheckBox5 End Dialog Dim d As UserDialog Dialog d End Sub </pre>
See Also	<p>DlgEnable (on page 434) (function); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).</p>

DlgFocus (function)

Syntax	DlgFocus\$[()]
--------	-----------------------

De- scrip- tion	Returns a String containing the name of the control with the focus.
Com- ments	The name of the control is the .Identifier parameter associated with the control in the dialog box template.
Exam- ple	<p>This code fragment makes sure that the control being disabled does not currently have the focus (otherwise, a runtime error would occur).</p> <pre> Sub Main() If DlgFocus = "Files" Then 'Does it have the focus? DlgFocus "OK" 'Change the focus to another control. End If DlgEnable "Files",False 'Now we can disable the control. End Sub </pre>
See Also	DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).

DlgFocus (statement)

Syn- tax	DlgFocus ControlName\$ ControllIndex
De- scrip- tion	Sets focus to the specified control.
Com- ments	A runtime error results if the specified control is hidden, disabled, or nonexistent. The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).

Example	<p>This code fragment makes sure the user enters a correct value. If not, the control returns focus back to the TextBox for correction.</p> <pre> Function DlgProc(ControlName\$,Action%,SuppValue%) As Integer If Action% = 2 and ControlName\$ = "OK" Then If IsNumeric(DlgText\$("TextBox1")) Then MsgBox "Duly Noted." Else MsgBox "Sorry, you must enter a number." DlgFocus "TextBox1" DlgProc = 1 End If End If End Function </pre>
	<pre> Sub Main() Dim ListBox1\$() Begin Dialog UserDialog , ,112,74,"Untitled" ,.DlgProc TextBox 12,20,88,12,..TextBox1 OKButton 12,44,40,14 CancelButton 60,44,40,14 Text 12,11,88,8,"Enter Desired Salary:",.Text1 End Dialog Dim d As UserDialog Dialog d End Sub </pre>
See Also	<p>DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).</p>

DlgListBoxArray (function)

Syntax	DlgListBoxArray ({ControlName\$ ControlIndex}, ArrayVariable)
--------	--

De- scrip- tion	Fills a list box, combo box, or drop list box with the elements of an array, returning an Integer containing the number of elements that were actually set into the control.
Com- ments	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
	The ArrayVariable parameter specifies a single-dimensional array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
Exam- ple	<p>This dialog function refills an array with files.</p> <pre> Function DlgProc(ControlName\$,Action%,SuppValue%) As Integer If Action% = 1 Then Dim NewFiles\$() 'Create a new dynamic array. FileList NewFiles\$,"c:*.*" 'Fill the array with files. r% = DlgListBoxArray("Files",NewFiles\$) 'Set items in the list box. DlgValue "Files",0 'Set the selection to the first item. DlgProc = 1 'Don't close the dialog box. End If End Function </pre>
	<pre> Sub Main() Dim ListBox1\$() Begin Dialog UserDialog , ,180,96,"Untitled",.DlgProc OKButton 132,8,40,14 CancelButton 132,28,40,14 ListBox 8,12,112,72,ListBox1\$,.Files End Dialog Dim d As UserDialog Dialog d End Sub </pre>
See Also	DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 439)

(statement); [DlgSetPicture \(on page 443\)](#) (statement); [DlgText \(on page 444\)](#) (statement); [DlgText\\$ \(on page 446\)](#) (function); [DlgValue \(on page 448\)](#) (function); [DlgValue \(on page 449\)](#) (statement); [DlgVisible \(on page 451\)](#) (statement); [DlgVisible \(on page 451\)](#) (function).

DlgListBoxArray (statement)

Syntax	DlgListBoxArray {ControlName\$ ControlIndex}, ArrayVariable
Description	Fills a list box, combo box, or drop list box with the elements of an array.
Comments	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControlIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
	The ArrayVariable parameter specifies a single-dimensional array used to initialize the elements of the control. If this array has no dimensions, then the control will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
Example	<p>This dialog function refills an array with files.</p> <pre> Function DlgProc(ControlName\$,Action\$,SuppValue%) As Integer If Action% = 1 Then Dim NewFiles\$() 'Create a new dynamic array. FileList NewFiles\$,"c:*.*" 'Fill the array with files. DlgListBoxArray "Files",NewFiles\$ 'Set items in the list box. DlgValue "Files",0 'Set the selection to the first item. DlgProc = 1 'Don't close the dialog box. End If End Function </pre> <pre> Sub Main() Dim ListBox1\$() Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc </pre>
	<pre> Sub Main() Dim ListBox1\$() Begin Dialog UserDialog ,,180,96,"Untitled",.DlgProc </pre>

	<pre> OKButton 132,8,40,14 CancelButton 132,28,40,14 ListBox 8,12,112,72,ListBox1\$, .Files End Dialog Dim d As UserDialog Dialog d End Sub </pre>
See Also	DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).

DlgProc (function)

Syn-tax	Function DlgProc(ControlName\$, Action, SuppValue) [As Integer]	
De-scrip-tion	Describes the syntax, parameters, and return value for dialog functions.	
Com-ments	Dialog functions are called by a script during the processing of a custom dialog box. The name of a dialog function (DlgProc) appears in the Begin Dialog statement as the .DlgProc parameter. Dialog functions require the following parameters:	
	Parameter	Description
	Control-Name\$	String containing the name of the control associated with Action.
	Action	Integer containing the action that called the dialog function.
	SuppValue	Integer of extra information associated with Action. For some actions, this parameter is not used.
	When a script displays a custom dialog box, the user may click on buttons, type text into edit fields, select items from lists, and perform other actions. When these actions occur, the Basic Control Engine calls the dialog function, passing it the action, the name of the control on which the action occurred, and any other relevant information associated with the action. The following table describes the different actions sent to dialog functions:	

Action	Description
1	<p>This action is sent immediately before the dialog box is shown for the first time. This gives the dialog function a chance to prepare the dialog box for use. When this action is sent, ControlName\$ contains a zero-length string, and SuppValue is 0. The return value from the dialog function is ignored in this case. Before Showing the Dialog Box After action 1 is sent, the Basic Control Engine performs additional processing before the dialog box is shown. Specifically, it cycles through the dialog box controls checking for visible picture or picture button controls. For each visible picture or picture button control, the Basic Control Engine attempts to load the associated picture. In addition to checking picture or picture button controls, the Basic Control Engine will automatically hide any control outside the confines of the visible portion of the dialog box. This prevents the user from tabbing to controls that cannot be seen. However, it does not prevent you from showing these controls with the <code>DlgVisible</code> statement in the dialog function.</p>
2	<p>This action is sent when:</p> <ul style="list-style-type: none"> • A button is clicked, such as OK, Cancel, or a push button. In this case, ControlName\$ contains the name of the button. SuppValue contains 1 if an OK button was clicked and 2 if a Cancel button was clicked; SuppValue is undefined otherwise. <p>If the dialog function returns 0 in response to this action, then the dialog box will be closed. Any other value causes the Basic Control Engine to continue dialog processing.</p> <ul style="list-style-type: none"> • A check box's state has been modified. In this case, ControlName\$ contains the name of the check box, and SuppValue contains the new state of the check box (1 if on, 0 if off). • An option button is selected. In this case, ControlName\$ contains the name of the option button that was clicked, and SuppValue contains the index of the option button within the option button group (0-based). • The current selection is changed in a list box, drop list box, or combo box. In this case, ControlName\$ contains the name of the list box, combo box, or drop list box, and SuppValue contains the index of the new item (0 is the first item, 1 is the second, and so on).
3	<p>This action is sent when the content of a text box or combo box has been changed. This action is only sent when the control loses focus. When this action is sent, ControlName\$</p>

	contains the name of the text box or combo box, and SuppValue contains the length of the new content. The dialog function's return value is ignored with this action.
4	This action is sent when a control gains the focus. When this action is sent, ControlName\$ contains the name of the control gaining the focus, and SuppValue contains the index of the control that lost the focus (0-based). The dialog function's return value is ignored with this action.
5	This action is sent continuously when the dialog box is idle. If the dialog function returns 1 in response to this action, then the idle action will continue to be sent. If the dialog function returns 0, then the Basic Control Engine will not send any additional idle actions. When the idle action is sent, ControlName\$ contains a zero-length string, and SuppValue contains the number of times the idle action has been sent so far. Note: Not returning zero will cause your application to use all available CPU time and may adversely affect your CIMPLICITY System.
6	This action is sent when the dialog box is moved. The ControlName\$ parameter contains a zero-length string, and SuppValue is 0. The dialog function's return value is ignored with this action.
	User-defined dialog boxes cannot be nested. In other words, the dialog function of one dialog box cannot create another user-defined dialog box. You can, however, invoke any built-in dialog box, such as MsgBox or InputDialog\$.
	<p>Within dialog functions, you can use the following additional statements and functions. These statements allow you to manipulate the dialog box controls dynamically.</p> <pre>DlgVisible DlgText\$ DlgText DlgSetPicture DlgListBoxArray DlgFocus DlgEnable DlgControlId</pre>
	The dialog function can optionally be declared to return a Variant . When returning a variable, the Basic Control Engine will attempt to convert the variant to an Integer . If the returned variant cannot be converted to an Integer , then 0 is assumed to be returned from the dialog function.
Example	<p>This dialog function enables/disables a group of option buttons when a check box is clicked.</p> <pre>Function SampleDlgProc(ControlName\$,Action%,SuppValue%) If Action% = 2 And ControlName\$ = "Printing" Then DlgEnable "PrintOptions",SuppValue% SampleDlgProc = 1 'Don't close the dialog box. End If End Function</pre>

```

Sub Main()

    Begin Dialog SampleDialogTemplate 34,39,106,45,"Sample",.SampleDlgProc

        OKButton 4,4,40,14

        CancelButton 4,24,40,14

        CheckBox 56,8,38,8,"Printing",.Printing

        OptionGroup .PrintOptions

            OptionButton 56,20,51,8,"Landscape",.Landscape

            OptionButton 56,32,40,8,"Portrait",.Portrait

        End Dialog

        Dim SampleDialog As SampleDialogTemplate

        SampleDialog.Printing = 1

        r% = Dialog(SampleDialog)

    End Sub

```

See [Begin Dialog \(on page 348\)](#) (statement).
Also

DlgSetPicture (statement)

Syn- tax	DlgSetPicture {ControlName\$ ControllIndex},PictureName\$,PictureType	
De- scrip- tion	Changes the content of the specified picture or picture button control.	
Com- ments	The DlgSetPicture statement accepts the following parameters:	
	Para- meter	Description
	Con- trol- Name\$	String containing the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specified control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
	Pic- ture- Name\$	String containing the name of the picture. If PictureType is 0, then this parameter specifies the name of the file containing the image. If PictureType is 10, then PictureName\$ specifies the name of the image within the resource of the picture library. If Picture-

		Name\$ is empty, then the current picture associated with the specified control will be deleted. Thus, a technique for conserving memory and resources would involve setting the picture to empty before hiding a picture control.
	Picture-Type	Integer specifying the source for the image. The following sources are supported:
	0	The image is contained in a file on disk.
	10	The image is contained in the picture library specified by the Begin Dialog statement. When this type is used, the PictureName\$ parameter must be specified with the Begin Dialog statement.
Example	<pre> Sub Main() DlgSetPicture "Picture1", "\windows\checks.bmp", 0 'Set picture from a file. DlgSetPicture 27, "FaxReport", 10 'Set control 10's image 'from a library. End Sub </pre>	
See Also	DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function), Picture (on page 657) (statement), PictureButton (on page 659) (statement).	
Notes	Picture controls can contain either bitmaps or WMFs (Windows metafiles). When extracting images from a picture library, the Basic Control Engine assumes that the resource type for metafiles is 256. Picture libraries are implemented as DLLs on the Windows and Win32 platforms.	

DlgText (statement)

Syntax	DlgText {ControlName\$ ControlIndex}, NewText\$
Description	Changes the text content of the specified control.

Comments	The effect of this statement depends on the type of the specified control:	
	Control Type	Effect of Dlg Text
	Picture	Runtime error.
	Option group	Runtime error.
	Drop list box	Sets the current selection to the item matching NewText\$. If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with NewText\$. If no match is found, then the selection is removed.
	OK button	Sets the label of the control to NewText\$.
	Cancel button	Sets the label of the control to NewText\$.
	Push button	Sets the label of the control to NewText\$.
	List box	Sets the current selection to the item matching NewText\$. If an exact match cannot be found, the DlgText statement searches from the first item looking for an item that starts with NewText\$. If no match is found, then the selection is removed.
	Combo box	Sets the content of the edit field of the combo box to NewText\$.
	Text	Sets the label of the control to NewText\$.
	Text box	Sets the content of the text box to NewText\$.

	Group box	Sets the label of the control to NewText\$.
	Option button	Sets the label of the control to NewText\$.
		The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
Example		<pre> Sub Main() DlgText "GroupBox1","Save Options" 'Change text of group box 1. If DlgText\$(9) = "Save Options" Then DlgText 9,"Editing Options" 'Change text to "Editing Options". End If End Sub </pre>
See Also		DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgValue (on page 449) (statement); DlgValue (on page 448) (function); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).

DlgText\$ (function)

Syntax	DlgText\$(ControlName\$ ControllIndex)
Description	Returns the text content of the specified control.
Comments	The text returned depends on the type of the specified control:

Control Type	Value Returned by DlgText\$
Picture	No value is returned. A runtime error occurs.
Option group	No value is returned. A runtime error occurs.
Drop list box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
OK button	Returns the label of the control.
Cancel button	Returns the label of the control.
Push button	Returns the label of the control.
List box	Returns the currently selected item. A zero-length string is returned if no item is currently selected.
Combo box	Returns the content of the edit field portion of the combo box.
Text	Returns the label of the control.
Text box	Returns the content of the control.
Group box	Returns the label of the control.
Option button	Returns the label of the control.
	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).
Example	<p>This code fragment makes sure the user enters a correct value. If not, the control returns focus back to the TextBox for correction.</p> <pre> Function DlgProc(ControlName\$,Action%,SuppValue%) As Integer If Action% = 2 and ControlName\$ = "OK" Then If IsNumeric(DlgText\$("TextBox1")) Then MsgBox "Duly Noted." </pre>

	<pre> Else MsgBox "Sorry, you must enter a number." DlgFocus "TextBox1" DlgProc = 1 End If End If End Function Sub Main() Dim ListBox1\$() Begin Dialog UserDialog , ,112,74,"Untitled",.DlgProc TextBox 12,20,88,12,.TextBox1 OKButton 12,44,40,14 CancelButton 60,44,40,14 Text 12,11,88,8,"Enter Desired Salary:",.Text1 End Dialog Dim d As UserDialog Dialog d End Sub </pre>
See Also	<p>DlgControlId (on page 431) (function); DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgValue (on page 448) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).</p>

DlgValue (function)

Syntax	DlgValue (ControlName\$ ControllIndex)
Description	Returns an Integer indicating the value of the specified control.
Comments	The value of any given control depends on its type, according to the following table:

	Control Type	DlgValue Returns
	Option group	The index of the selected option button within the group (0 is the first option button, 1 is the second, and so on).
	List box	The index of the selected item.
	Drop list box	The index of the selected item.
	Check box	1 if the check box is checked; 0 otherwise.
	A runtime error is generated if DlgValue is used with controls other than those listed in the above table. The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).	
Example	<p>This code fragment toggles the value of a check box.</p> <pre data-bbox="305 1010 682 1304"> Sub Main() If DlgValue("MyCheckBox") = 1 Then DlgValue "MyCheckBox",0 Else DlgValue "MyCheckBox",1 End If End Sub </pre>	
See Also	DlgControlId (on page 431) (function); DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 449) (statement); DlgVisible (on page 451) (statement); DlgVisible (on page 451) (function).	

DlgValue (statement)

Syntax	DlgValue {ControlName\$ ControllIndex},Value
--------	---

De- scrip- tion	Changes the value of the given control.	
Com- ments	The value of any given control is an Integer and depends on its type, according to the following table:	
	Control Type	Description of Value
	Option group	The index of the new selected option button within the group (0 is the first option button, 1 is the second, and so on).
	List box	The index of the new selected item.
	Drop list box	The index of the new selected item.
	Check box	1 if the check box is to be checked; 0 if the check is to be removed.
	A runtime error is generated if DlgValue is used with controls other than those listed in the above table.	
	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).	
Exam- ple	<p>This code fragment toggles the value of a check box.</p> <pre> Sub Main() If DlgValue("MyCheckBox") = 1 Then DlgValue "MyCheckBox",0 Else DlgValue "MyCheckBox",1 End If End Sub </pre>	
See Also	DlgControlId (on page 431) (function); DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on	

[page 446](#)) (function); [DlgValue \(on page 448\)](#) (function); [DlgVisible \(on page 451\)](#) (statement); [DlgVisible \(on page 451\)](#) (function).

DlgVisible (function)

Syn-tax	DlgVisible (ControlName\$ ControllIndex)
De-scrip-tion	Returns True if the specified control is visible; returns False otherwise .
	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the template (0 is the first control in the template, 1 is the second, and so on). A runtime error is generated if DlgVisible is called with no user dialog is active.
Ex-ample	<pre>Sub Main() If DlgVisible("Portrait") Then Beep If DlgVisible(10) And DlgVisible(12) Then MsgBox "The 10th and 12th controls are visible." End If End Sub</pre>
See Also	DlgControlId (on page 431) (function); DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 449) (statement); DlgValue (on page 449) (statement); DlgVisible (on page 451) (function).

DlgVisible (statement)

Syn-tax	DlgVisible {ControlName\$ ControllIndex} [,isOn]
---------	---

De- scrip- tion	Hides or shows the specified control.	
Com- ments	Hidden controls cannot be seen in the dialog box and cannot receive the focus using Tab. The isOn parameter is an Integer specifying the new state of the control. It can be any of the following values:	
	1	The control is shown.
	0	The control is hidden.
	Omitted Toggles the visibility of the control. Option buttons can be manipulated individually (by specifying an individual option button) or as a group (by specifying the name of the option group).	
	The ControlName\$ parameter contains the name of the .Identifier parameter associated with a control in the dialog box template. A case-insensitive comparison is used to locate the specific control within the template. Alternatively, by specifying the ControllIndex parameter, a control can be referred to using its index in the dialog box template (0 is the first control in the template, 1 is the second, and so on).	
	Picture Caching When the dialog box is first created and before it is shown, the Basic Control Engine calls the dialog function with action set to 1. At this time, no pictures have been loaded into the picture controls contained in the dialog box template. After control returns from the dialog function and before the dialog box is shown, the Basic Control Engine will load the pictures of all visible picture controls. Thus, it is possible for the dialog function to hide certain picture controls, which prevents the associated pictures from being loaded and causes the dialog box to load faster. When a picture control is made visible for the first time, the associated picture will then be loaded.	
Exam- ple	<p>This example creates a dialog box with two panels. The DlgVisible statement is used to show or hide the controls of the different panels.</p> <pre data-bbox="305 1514 1427 1871"> Sub EnableGroup(start%,finish%) For i = 6 To 13 'Disable all options. DlgVisible i,False Next i For i = start% To finish% 'Enable only the right ones. DlgVisible i,True Next i End Sub </pre>	

```

Function DlgProc(ControlName$,Action%,SuppValue%)

  If Action% = 1 Then

    DlgValue "WhichOptions",0 'Set to save options.

    EnableGroup 6,8 'Enable the save options.

  End If

  If Action% = 2 And ControlName$ = "SaveOptions" Then

    EnableGroup 6,8 'Enable the save options.

    DlgProc = 1 'Don't close the dialog box.

  End If

  If Action% = 2 And ControlName$ = "EditingOptions" Then

    EnableGroup 9,13 'Enable the editing options.

    DlgProc = 1 'Don't close the dialog box.

  End If

End Function

Sub Main()

  Begin Dialog OptionsTemplate 33,33,171,134,"Options",.DlgProc

    'Background (controls 0-5)

    GroupBox 8,40,152,84,""

    OptionGroup .WhichOptions

      OptionButton 8,8,59,8,"Save Options",.SaveOptions

      OptionButton 8,20,65,8,"Editing Options",.EditingOptions

    OKButton 116,7,44,14

    CancelButton 116,24,44,14

    'Save options (controls 6-8)

    CheckBox 20,56,88,8,"Always create backup",.CheckBox1

    CheckBox 20,68,65,8,"Automatic save",.CheckBox2

    CheckBox 20,80,70,8,"Allow overwriting",.CheckBox3

    'Editing options (controls 9-13)

    CheckBox 20,56,65,8,"Overtyp e mode",.Overtyp eMode

    CheckBox 20,68,69,8,"Uppercase only",.UppercaseOnly

    CheckBox 20,80,105,8,"Automatically check syntax",.AutoCheckSyntax

    CheckBox 20,92,73,8,"Full line selection",.FullLineSelection

    CheckBox 20,104,102,8,"Typing replaces selection",.TypingReplacesText

  End Dialog

```

	<pre>Dim OptionsDialog As OptionsTemplate Dialog OptionsDialog End Sub</pre>
See Also	DlgControlId (on page 431) (function); DlgEnable (on page 432) (function); DlgEnable (on page 434) (statement); DlgFocus (on page 435) (function); DlgFocus (on page 436) (statement); DlgListBoxArray (on page 437) (function); DlgListBoxArray (on page 439) (statement); DlgSetPicture (on page 443) (statement); DlgText (on page 444) (statement); DlgText\$ (on page 446) (function); DlgValue (on page 449) (statement); DlgValue (on page 448) (function); DlgVisible (on page 451) (statement).

Do...Loop (statement)

Syn-tax 1	Do {While Until} condition statements Loop
Syn-tax 2	Do statements Loop {While Until} condition
Syn-tax 3	Do statements Loop
Description	Repeats a block of Basic Control Engine statements while a condition is True or until a condition is True .
Comments	If the {While Until} conditional clause is not specified, then the loop repeats the statements forever (or until the script encounters an Exit Do statement). The condition parameter specifies any Boolean expression.
Examples	<pre>Sub Main() 'This first example uses the Do...While statement, which performs 'the iteration, then checks the condition, and repeats if the 'condition is True. Dim a\$(100) i% = -1 Do i% = i% + 1 If i% = 0 Then a(i%) = Dir("***")</pre>

```

Else
    a(i%) = Dir
End If

Loop While(a(i%) <> "" And i% <= 99)

r% = SelectBox(i% & " files found",,a)

End Sub

Sub Main()

'This second example uses the Do While...Loop, which checks the
'condition and then repeats if the condition is True.

Dim a$(100)

i% = 0

a(i%) = Dir("")

Do While (a(i%) <> "") And (i% <= 99)

    i% = i% + 1

    a(i%) = Dir

Loop

r% = SelectBox(i% & " files found",,a)

End Sub

Sub Main()

'This third example uses the Do Until...Loop, which does the
'iteration and then checks the condition and repeats if the
'condition is True.

Dim a$(100)

i% = 0

a(i%) = Dir("")

Do Until (a(i%) = "") Or (i% = 100)

    i% = i% + 1

    a(i%) = Dir

Loop

r% = SelectBox(i% & " files found",,a)

End Sub

Sub Main()

'This last example uses the Do...Until Loop, which performs the
'iteration first, checks the condition, and repeats if the
'condition is True.

```


	<pre> Dim a\$(100) i% = -1 Do i% = i% + 1 If i% = 0 Then a(i%) = Dir("**") Else a(i%) = Dir End If Loop Until (a(i%) = "") Or (i% = 100) r% = SelectBox(i% & " files found",,a) End Sub </pre>
See Also	For...Next (on page 524) (statement); While ...WEnd (on page 780) (statement).
Notes:	Due to errors in program logic, you can inadvertently create infinite loops in your code. You can break out of infinite loops using Ctrl+Break.

DoEvents (function)

Syntax	DoEvents[()]
Description	Yields control to other applications, returning an Integer 0 .
Comments	This statement yields control to the operating system, allowing other applications to process mouse, keyboard, and other messages. If a SendKeys statement is active, this statement waits until all the keys in the queue have been processed.
Example	<p>The following routine explicitly yields to allow other applications to execute and refresh on a regular basis.</p> <pre> Sub Main() Open "test.txt" For Output As #1 For i = 1 To 10000 Print #1,"This is a test of the system and such." r = DoEvents End For End Sub </pre>

	<pre> Next i MsgBox "The DoEvents return value is: " & r Close #1 End Sub </pre>
See Also	DoEvents (on page 457) (statement).

DoEvents (statement)

Syn-tax	DoEvents
De-scrip-tion	Yields control to other applications.
Com-ments	This statement yields control to the operating system, allowing other applications to process mouse, keyboard, and other messages. If a SendKeys statement is active, this statement waits until all the keys in the queue have been processed.
Exam-ples	<p>This first example shows a script that takes a long time and hogs the system. The following routine explicitly yields to allow other applications to execute and refresh on a regular basis.</p> <pre> Sub Main() Open "test.txt" For Output As #1 For i = 1 To 10000 Print #1,"This is a test of the system and stuff." DoEvents Next i Close #1 End Sub </pre>
	<p>In this second example, the DoEvents statement is used to wait until the queue has been completely flushed.</p> <pre> Sub Main() id = Shell("notepad.exe",3) 'Start new instance of Notepad. SendKeys "This is a test.",False 'Send some keys. oEvents 'Wait for the keys to play back. End Sub </pre>

See	DoEvents (on page 457) (statement).
Also	

Double (data type)

Syn-tax	Double	
De-scrip-tion	A data type used to declare variables capable of holding real numbers with 15–16 digits of precision.	
Com-ments	Double variables are used to hold numbers within the following ranges:	
	Sign	Range
	Negative	– 1.797693134862315E308 <= double <=
		-4.94066E-324
	Positive	4.94066E-324 <= double <= 1.797693134862315E308
	<p>The type-declaration character for Double is #. Storage</p> <ul style="list-style-type: none"> Internally, doubles are 8-byte (64-bit) IEEE values. Thus, when appearing within a structure, doubles require 8 bytes of storage. When used with binary or random files, 8 bytes of storage are required. 	
	<p>Each Double consists of the following</p> <ul style="list-style-type: none"> A 1-bit sign An 11-bit exponent A 53-bit significand (mantissa) 	
See	Currency (on page 387) (data type); Date (on page 392) (data type); Integer (on page 566)	
Also	(data type) ; Long (on page 598) (data type); Object (on page 633) (data type); Single (on page 718) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement); CDbl (on page 356) (function).	

DropListBox (statement)

Syntax	DropListBox X, Y, width, height, ArrayVariable, .Identifier	
Description	Creates a drop list box within a dialog box template.	
Comments	<p>When the dialog box is invoked, the drop list box will be filled with the elements contained in ArrayVariable. Drop list boxes are similar to combo boxes, with the following exceptions:</p> <ul style="list-style-type: none"> • The list box portion of a drop list box is not opened by default. The user must open it by clicking the down arrow. • The user cannot type into a drop list box. Only items from the list box may be selected. With combo boxes, the user can type the name of an item from the list directly or type the name of an item that is not contained within the combo box. <p>This statement can only appear within a dialog box template (i.e., between the Begin Dialog and End Dialog statements). The <code>DropListBox</code> statement requires the following parameters:</p>	
	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Array-Variable	Single-dimensioned array used to initialize the elements of the drop list box. If this array has no dimensions, then the drop list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension.
		ArrayVariable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
	.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the drop list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax:
		DialogVariable.Identifier
Example	This example allows the user to choose a field name from a drop list box.	

```

Sub Main()

    Dim FieldNames$(4)

    FieldNames$(0) = "Last Name"

    FieldNames$(1) = "First Name"

    FieldNames$(2) = "Zip Code"

    FieldNames$(3) = "State"

    FieldNames$(4) = "City"

    Begin Dialog FindTemplate 16,32,168,48,"Find"

        Text 8,8,37,8,"&Find what:"

        DropDownList 48,6,64,80,FieldNames,.WhichField

        OKButton 120,7,40,14

        CancelButton 120,27,40,14

    End Dialog

    Dim FindDialog As FindTemplate

    FindDialog.WhichField = 1

    Dialog FindDialog

End Sub

```

See [CancelButton \(on page 365\)](#) (statement); [CheckBox \(on page 360\)](#) (statement); [ComboBox \(on page 373\)](#) (statement); [Dialog \(on page 424\)](#) (function); [Dialog \(on page 426\)](#) (statement); [GroupBox \(on page 544\)](#) (statement); [ListBox \(on page 591\)](#) (statement); [OKButton \(on page 638\)](#) (statement); [OptionButton \(on page 652\)](#) (statement); [OptionGroup \(on page 653\)](#) (statement); [Picture \(on page 657\)](#) (statement); [PushButton \(on page 671\)](#) (statement); [Text \(on page 752\)](#) (statement); [TextBox \(on page 753\)](#) (statement); [Begin \(on page 348\)](#) [Dialog \(on page 348\)](#) (statement), [PictureButton \(on page 659\)](#) (statement).

E

E

ebAbort (constant)
ebAbortRetryIgnore (constant)
ebApplicationModal (constant)
ebArchive (constant)
ebBold (constant)

ebBoldItalic (constant)
ebBoolean (constant)
ebCancel (constant)
ebCritical (constant)
ebCurrency (constant)
ebDataObject (constant)
ebDate (constant)
ebDefaultButton1 (constant)
ebDefaultbutton2 (constant)
ebDefaultbutton3 (constant)
ebDirectory (constant)
ebDos (constant)
ebDouble (constant)
ebEmpty (constant)
ebError (constant)
ebExclamation (constant)
ebHidden (constant)
ebIgnore (constant)
ebInformation (constant)
ebInteger (constant)
ebItalic (constant)
ebLong (constant)
ebNo (constant)
ebNone (constant)
ebNormal (constant)
ebNull (constant)
ebObject (constant)

ebOK (constant)
ebOKCancel (constant)
ebOKOnly (constant)
ebQuestion (constant)
ebReadOnly (constant)
ebRegular (constant)
ebRetry (constant)
ebRetryCancel (constant)
ebSingle (constant)
ebString (constant)
ebSystem (constant)
ebSystemModal (constant)
ebVariant (constant)
ebVolume (constant)
ebYes (constant)
ebYesNo (constant)
ebYesNoCancel (constant)
Empty (constant)
End (statement)
End Dialog (statement)
Environ, Environ\$ (function)
EOF (function)
Eqv (operator)
Erase (statement)
Erl (function)
Err (function) obsolete
Err (statement)

Err.Clear (method)
Err.Description (property)
Err.HelpContext (property)
Err.HelpFile (property)
Err.LastDLLError (property)
Err.Number (property)
Err.Raise (method)
Err.Source (property)
Error (statement)
Error Handling (topic)
Error, Error\$ ((functions)
Exit Do (statement)
Exit For (statement)
Exit Function (statement)
Exit Sub (statement)
Exp (function)
Expression Evaluation (topic)

ebAbort (constant)

Description	Returned by the MsgBox function when the Abort button is chosen.
Comments	This constant is equal to 3.
Example	<p>This example displays a dialog box with Abort, Retry, and Ignore buttons.</p> <pre> Sub Main() Again: rc% = MsgBox("Do you want to continue?",ebAbortRetryIgnore) If rc% = ebAbort or rc% = ebIgnore Then End </pre>

	<pre> ElseIf rc% = ebRetry Then Goto Again End If End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebAbortRetryIgnore (constant)

Description	Used by the MsgBox statement and function.
Comments	This constant is equal to 2.
Example	<p>This example displays a dialog box with Abort, Retry, and Ignore buttons.</p> <pre> Sub Main() Again: rc% = MsgBox("Do you want to continue?", ebAbortRetryIgnore) If rc% = ebAbort or rc% = ebIgnore Then End ElseIf rc% = ebRetry Then Goto Again End If End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebApplicationModal (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 0.
Example	This example displays an application-modal dialog box (which is the default).

	<pre>Sub Main() MsgBox "This is application-modal.", vbOKOnly Or vbApplicationModal End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebArchive (constant)

Description	Bit position of a file attribute indicating that a file hasn't been backed up.
Comments	This constant is equal to 32.
Example	<p>This example dimensions an array and fills it with filenames with the Archive bit set.</p> <pre>Sub Main() Dim s\$() FileList s\$, "*", ebArchive a% = SelectBox("Archived Files", "Choose one", s\$()) If a% >= 0 Then 'If a% is -1, then the user pressed Cancel. MsgBox "You selected Archive file: " & s\$(a) Else MsgBox "No selection made." End If End Sub</pre>
See Also	Dir, Dir\$(on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebBold (constant)

Description	Used with the Text and TextBox statement to specify a bold font.
Comments	This constant is equal to 2.
Example	<pre>Sub Main() Begin Dialog UserDialog 16,32,232,132,"Bold Font Demo" Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBold TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebBold End Dialog</pre>

	<pre> OKButton 96,110,40,14 End Dialog Dim a As UserDialog Dialog a End Sub </pre>
See Also	Text (on page 752) (statement), TextBox (on page 753) (statement).

ebBoldItalic (constant)

Description	Used with the Text and TextBox statement to specify a bold-italic font.
Comments	This constant is equal to 6.
Example	<pre> Sub Main() Begin Dialog UserDialog 16,32,232,132,"Bold-Italic Font Demo" Text 10,10,200,20,"Hello, world.",,"Helv",24,ebBoldItalic TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebBoldItalic OKButton 96,110,40,14 End Dialog Dim a As UserDialog Dialog a End Sub </pre>
See Also	Text (on page 752) (statement), TextBox (on page 753) (statement).

ebBoolean (constant)

Description	Number representing the type of a Boolean variant.
Comments	This constant is equal to 11.
Example	<pre> Sub Main() Dim MyVariant as variant MyVariant = True If VarType(MyVariant) = ebBoolean Then </pre>

	<pre> MyVariant = 5.5 End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebCancel (constant)

Description	Returned by the MsgBox function when the Cancel button is chosen.
Comments	This constant is equal to 2.
Example	<pre> Sub Main() 'Invoke MsgBox and check whether the Cancel button was pressed. rc% = MsgBox("Are you sure you want to quit?",ebOKCancel) If rc% = ebCancel Then MsgBox "The user clicked Cancel." End If End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebCritical (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 16.
Example	<pre> Sub Main() 'Invoke MsgBox with Abort, Retry, and Ignore buttons and a Stop icon. rc% = MsgBox("Disk drive door is open.",ebAbortRetryIgnore Or ebCritical) If rc% = 3 Then 'The user selected Abort from the dialog box. MsgBox "The user clicked Abort." End If End Sub </pre>

See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).
----------	--

ebCurrency (constant)

Description	Number representing the type of a Currency variant.
Comments	This constant is equal to 6.
Example	<p>This example checks to see whether a variant is of type Currency.</p> <pre> Sub Main() Dim MyVariant If VarType(MyVariant) = ebCurrency Then MsgBox "Variant is Currency." End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebDataObject (constant)

Description	Number representing the type of a data object variant.
Comments	This constant is equal to 13.
Example	<p>This example checks to see whether a variable is a data object.</p> <pre> Sub Main() Dim MyVariant as Variant If VarType(MyVariant) = ebDataObject Then MsgBox "Variant contains a data object." End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebDate (constant)

Description	Number representing the type of a Date variant.
Comments	This constant is equal to 7.
Example	<pre>Sub Main() Dim MyVariant as Variant If VarType(MyVariant) = ebDate Then MsgBox "This variable is a Date type!" Else MsgBox "This variable is not a Date type!" End If End Sub</pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebDefaultButton1 (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 0.
Example	<p>This example invokes MsgBox with the focus on the OK button by default.</p> <pre>Sub Main() rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or ebDefaultButton1) End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebDefaultButton2 (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 256.
Example	<p>This example invokes MsgBox with the focus on the Cancel button by default.</p>

	<pre>Sub Main() rc% = MsgBox("Are you sure you want to quit?",ebOKCancel Or ebDefaultButton2) End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebDefaultButton3 (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 512.
Example	<p>This example invokes MsgBox with the focus on the Ignore button by default.</p> <pre>Sub Main() rc% = MsgBox("Disk drive door open.",ebAbortRetryIgnore Or ebDefaultButton3) End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebDirectory (constant)

Description	Bit position of a file attribute indicating that a file is a directory entry.
Comments	This constant is equal to 16.
Example	<p>This example dimensions an array and fills it with directory names using the ebDirectory constant.</p> <pre>Sub Main() Dim s\$() FileList s\$,"c:*",ebDirectory a% = SelectBox("Directories", "Choose one:", s\$) If a% >= 0 Then MsgBox "You selected directory: " & s(a%) Else MsgBox "No selection made."</pre>

	<pre>End If End Sub</pre>
See Also	Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebDos (constant)

Description	Used with the AppType or FileType functions to indicate a DOS application.
Comments	This constant is equal to 1.
Example	<p>This example detects whether a DOS program was selected.</p> <pre>Sub Main() s\$ = OpenFilename\$("Run", "Programs:*.*") If s\$ <> "" Then If FileType(s\$) = ebDos Then MsgBox "You selected a DOS exe file." End If End If End Sub</pre>
See Also	AppType (on page 327) (function)

ebDouble (constant)

Description	Number representing the type of a Double variant.
Comments	This constant is equal to 5.
Example	See ebSingle (constant).
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement); VarType (on page 773) (function); Variant (on page 771) (data type).

ebEmpty (constant)

Description	Number representing the type of an Empty variant.
Comments	This constant is equal to 0.

Example	<pre> Sub Main() Dim MyVariant as Variant If VarType(MyVariant) = ebEmpty Then MsgBox "This variant has not been assigned a value yet!" End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebError (constant)

Description	Number representing the type of an error variant.
Comments	This constant is equal to 10.
Example	<p>This example checks to see whether a variable is an error.</p> <pre> Function Div(ByVal a As Variant,ByVal b As Variant) As Variant On Error Resume Next Div = a / b If Err <> 0 Then Div = CVErr(Err) End Function Sub Main() a = InputBox("Please enter 1st number","Division Sample") b = InputBox("Please enter 2nd number","Division Sample") res = Div(a,b) If VarType(res) = ebError Then res = CStr(res) res = Error(Mid(res,7,Len(res))) MsgBox "" & res & "" occurred" Else MsgBox "The result of the division is: " & res End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebExclamation (constant)

Description	Used with the <code>MsgBox</code> statement and function.
Comments	This constant is equal to 48.
Example	<p>This example displays a dialog box with an OK button and an exclamation icon.</p> <pre> Sub Main() MsgBox "Out of memory saving to disk.",ebOKOnly Or ebExclamation End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebHidden (constant)

Description	Bit position of a file attribute indicating that a file is hidden.
Comments	This constant is equal to 2.
Example	<p>This example dimensions an array and fills it with filenames using the <code>ebHidden</code> attribute.</p> <pre> Sub Main() Dim s\$() FileList s\$,"*",ebHidden If ArrayDims(s\$) = 0 Then MsgBox "No hidden files found!" End End If a% = SelectBox("Hidden Files","Choose one", s\$) If a% >= 0 Then MsgBox "You selected hidden file " & s(a%) Else MsgBox "No selection made." End If End Sub </pre>
See Also	Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebIgnore (constant)

Description	Returned by the MsgBox function when the Ignore button is chosen.
Comments	This constant is equal to 5.
Example	<p>This example displays a critical error dialog box and sees what the user wants to do.</p> <pre> Sub Main() rc% = MsgBox("Printer out of paper.",ebAbortRetryIgnore) If rc% = ebIgnore Then 'Continue printing here. End If End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebInformation (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 64.
Example	<p>This example displays a dialog box with the Information icon.</p> <pre> Sub Main() MsgBox "You just deleted your file!",ebOKOnly Or ebInformation End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebInteger (constant)

Description	Number representing the type of an Integer variant.
Comments	This constant is equal to 2.

Example	<p>This example defines a function that returns True if a variant contains an Integer value (either a 16-bit or 32-bit Integer).</p> <pre> Function IsInteger(v As Variant) As Boolean If VarType(v) = ebInteger Or VarType(v) = ebLong Then IsInteger = True Else IsInteger = False End If End Function Sub Main() Dim i as Integer i = 123 If IsInteger(i) then MsgBox "i is an Integer." End If End Sub </pre>
See Also	<p>VarType (on page 773) (function); Variant (on page 771) (data type).</p>

ebItalic (constant)

Description	Used with the Text and TextBox statement to specify an italic font.
Comments	This constant is equal to 4.
Example	<pre> Sub Main() Begin Dialog UserDialog 16,32,232,132,"Italic Font Demo" Text 10,10,200,20,"Hello, world.",,"Helv",24,ebItalic TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebItalic OKButton 96,110,40,14 End Dialog Dim a As UserDialog Dialog a End Sub </pre>
See Also	<p>Text (on page 752) (statement), TextBox (on page 753) (statement).</p>

ebLong (constant)

Description	Number representing the type of a Long variant.
Comments	This constant is equal to 3.
Example	See ebInteger (constant).
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebNo (constant)

Description	Returned by the MsgBox function when the No button is chosen.
Comments	This constant is equal to 7.
Example	<p>This example asks a question and queries the user's response.</p> <pre> Sub Main() rc% = MsgBox("Do you want to update the glossary?",ebYesNo) If rc% = ebNo Then MsgBox "The user clicked 'No'." 'Don't update glossary. End If End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebNone (constant)

Description	Bit value used to select files with no other attributes.
Comments	This value can be used with the <code>Dir\$</code> and <code>FileList</code> commands. These functions will return only files with no attributes set when used with this constant. This constant is equal to 64.
Example	This example dimensions an array and fills it with filenames with no attributes set.

	<pre> Sub Main() Dim s\$() FileList s\$,"*",ebNone If ArrayDims(s\$) = 0 Then MsgBox "No files found without attributes!" End End If a% = SelectBox("No Attributes", "Choose one", s\$) If a% >= 0 Then MsgBox "You selected file " & s(a%) Else MsgBox "No selection made." End If End Sub </pre>
See Also	Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebNormal (constant)

De- scrip- tion	Used to search for normal files.
Com- ments	This value can be used with the <code>Dir\$</code> and <code>FileList</code> commands and will return files with the Archive, Volume, ReadOnly, or no attributes set. It will not match files with Hidden, System, or Directory attributes. This constant is equal to 0.
Exam- ple	<p>This example dimensions an array and fills it with filenames with Normal attributes.</p> <pre> Sub Main() Dim s\$() FileList s\$,"*", ebNormal If ArrayDims(s\$) = 0 Then MsgBox "No filesfound!" End End If a% = SelectBox("Normal Files", "Choose one", s\$) If a% >= 0 Then </pre>

	<pre> MsgBox "You selected file " & s(a%) Else MsgBox "No selection made." End If End Sub </pre>
See Also	Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebNull (constant)

Description	Number representing the type of a Null variant.
Comments	This constant is equal to 1.
Example	<pre> Sub Main() Dim MyVariant MyVariant = Null If VarType(MyVariant) = ebNull Then MsgBox "This variant is Null" End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebObject (constant)

Description	Number representing the type of an Object variant (an OLE automation object).
Comments	This constant is equal to 9.
Example	<pre> Sub Main() Dim MyVariant If VarType(MyVariant) = ebObject Then MsgBox MyVariant.Value Else MsgBox "'MyVariant' is not an object." End If End Sub </pre>

	<pre>End If End Sub</pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebOK (constant)

Description	Returned by the MsgBox function when the OK button is chosen.
Comments	This constant is equal to 1.
Example	<p>This example displays a dialog box that allows the user to cancel.</p> <pre>Sub Main() rc% = MsgBox("Are you sure you want to exit Windows?", ebOKCancel) If rc% = ebOK Then System.Exit End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebOKCancel (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 1.
Example	<p>This example displays a dialog box that allows the user to cancel.</p> <pre>Sub Main() rc% = MsgBox("Are you sure you want to exit Windows?", ebOKCancel) If rc% = ebOK Then System.Exit End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebOKOnly (constant)

Description	Used with the MsgBox statement and function
Comments	This constant is equal to 0.
Example	This example informs the user of what is going on (no options). <pre>Sub Main() MsgBox "The system has been reset.", vbOKOnly End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement)

vbQuestion (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 32.
Example	This example displays a dialog box with OK and Cancel buttons and a question icon. <pre>Sub Main() rc% = MsgBox("OK to delete file?", vbOKCancel Or vbQuestion) End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement)

vbReadOnly (constant)

Description	Bit position of a file attribute indicating that a file is read-only.
Comments	This constant is equal to 1.
Example	This example dimensions an array and fills it with filenames with ReadOnly attributes. <pre>Sub Main() Dim s\$() FileList s\$, "*", vbReadOnly If ArrayDims(s\$) = 0 Then MsgBox "No read only files found!" End If End Sub</pre>

<pre> End End If a% = SelectBox("ReadOnly", "Choose one", s%) If a% >= 0 Then MsgBox "You selected file " & s(a%) Else MsgBox "No selection made." End If End Sub </pre>
<p>See Also Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).</p>

ebRegular (constant)

Description	Used with the Text and TextBox statement to specify a normal-styled font (i.e., neither bold or italic).
Comments	This constant is equal to 1.
Example	<pre> Sub Main() Begin Dialog UserDialog 16,32,232,132,"Regular Font Demo" Text 10,10,200,20,"Hello, world.",,"Helv",24,ebRegular TextBox 10,35,200,20,.Edit,,"Times New Roman",16,ebRegular OKButton 96,110,40,14 End Dialog Dim a As UserDialog Dialog a End Sub </pre>
See Also	Text (on page 752) (statement), TextBox (on page 753) (statement)

ebRetry (constant)

Description	Returned by the MsgBox function when the Retry button is chosen.
Comments	This constant is equal to 4.

Example	<p>This example displays a Retry message box.</p> <pre> Sub Main() rc% = MsgBox("Unable to open file.",ebRetryCancel) If rc% = ebRetry Then MsgBox "User selected Retry." End If End Sub </pre>
See Also	<p>MsgBox (on page 614) (function); MsgBox (on page 617) (statement)</p>

ebRetryCancel (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 5.
Example	<p>This example invokes a dialog box with Retry and Cancel buttons.</p> <pre> Sub Main() rc% = MsgBox("Unable to open file.",ebRetryCancel) End Sub </pre>
See Also	<p>MsgBox (on page 614) (function); MsgBox (on page 617) (statement).</p>

ebSingle (constant)

Description	Number representing the type of a Single variant.
Comments	This constant is equal to 4.
Example	<p>This example defines a function that returns True if the passed variant is a Real number.</p> <pre> Function IsReal(v As Variant) As Boolean If VarType(v) = ebSingle Or VarType(v) = ebDouble Then IsReal = True Else IsReal = False End If End Function </pre>

```

End If
End Function
Sub Main()
    Dim i as Integer
    i = 123
    If IsReal(i) then
        MsgBox "i is Real."
    End If
End Sub

```

See Also [VarType \(on page 773\)](#) (function); [Variant \(on page 771\)](#) (data type).

ebString (constant)

Description	Number representing the type of a String variant.
Comments	This constant is equal to 8.
Example	<pre> Sub Main() Dim MyVariant as variant MyVariant = "This is a test." If VarType(MyVariant) = ebString Then MsgBox "Variant is a string." End If End Sub </pre>
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebSystem (constant)

Description	Bit position of a file attribute indicating that a file is a system file.
Comments	This constant is equal to 4.
Example	This example dimensions an array and fills it with filenames with System attributes.

	<pre> Sub Main() Dim s\$() FileList s\$,"*",ebSystem a% = SelectBox("System Files", "Choose one", s\$) If a% >= 0 Then MsgBox "You selected file " & s(a%) Else MsgBox "No selection made." End If End Sub </pre>
See Also	Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebSystemModal (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 4096.
Example	<pre> Sub Main() MsgBox "All applications are halted!",ebSystemModal End Sub </pre>
See Also	ebApplicationModal (on page 464) (constant); Constants (topic); MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebVariant (constant)

Description	Number representing the type of a Variant .
Comments	Currently, it is not possible for variants to use this subtype. This constant is equal to 12.
See Also	VarType (on page 773) (function); Variant (on page 771) (data type).

ebVolume (constant)

Description	Bit position of a file attribute indicating that a file is the volume label.
Comments	This constant is equal to 8.
Example	<p>This example dimensions an array and fills it with filenames with Volume attributes.</p> <pre> Sub Main() Dim s\$() FileList s\$, "*", ebVolume If ArrayDims(s\$) > 0 Then MsgBox "The volume name is: " & s(1) Else MsgBox "No volumes found." End If End Sub </pre>
See Also	Dir, Dir\$ (on page 427) (functions); FileList (on page 517) (statement); SetAttr (on page 715) (statement); GetAttr (on page 538) (function); FileAttr (on page 512) (function).

ebYes (constant)

Description	Returned by the MsgBox function when the Yes button is chosen.
Comments	This constant is equal to 6.
Example	<p>This example queries the user for a response.</p> <pre> Sub Main() rc% = MsgBox("Overwrite file?", ebYesNoCancel) If rc% = ebYes Then MsgBox "You elected to overwrite the file." End If End Sub </pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebYesNo (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 4.
Example	<p>This example displays a dialog box with Yes and No buttons.</p> <pre>Sub Main() rc% = MsgBox("Are you sure you want to remove all formatting?",ebYesNo) End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

ebYesNoCancel (constant)

Description	Used with the MsgBox statement and function.
Comments	This constant is equal to 3.
Example	<p>This example displays a dialog box with Yes, No, and Cancel buttons.</p> <pre>Sub Main() rc% = MsgBox("Format drive C:?",ebYesNoCancel) If rc% = ebYes Then MsgBox "The user chose Yes." End If End Sub</pre>
See Also	MsgBox (on page 614) (function); MsgBox (on page 617) (statement).

Empty (constant)

De- scrip- tion	Constant representing a variant of type 0.
Com- ments	The Empty value has special meaning indicating that a Variant is uninitialized. When Empty is assigned to numbers, the value 0 is assigned. When Empty is assigned to a String , the string is assigned a zero-length string.

Example	<pre>Sub Main() Dim a As Variant a = Empty MsgBox "This string is" & a & "concatenated with Empty" MsgBox "5 + Empty = " & (5 + a) End Sub</pre>
See Also	Null (on page 632) (constant); Variant (on page 771) (data type); VarType (on page 773) (function).

End (statement)

Syntax	End
Description	Terminates execution of the current script, closing all open files.
Example	<p>This example uses the End statement to stop execution.</p> <pre>Sub Main() MsgBox "The next line will terminate the script." End End Sub</pre>
See Also	Close (on page 373) (statement); Stop (on page 737) (statement); Exit For (on page 506) (statement); Exit Do (on page 505) (statement); Exit Function (on page 507) (statement); Exit Sub (on page 507) (function).

End Dialog (statement)

Syntax	Begin Dialog (on page 348) DialogName [x],[y],width,height,title\$ [,[.DlgProc] [,[PicName\$] [,[style]]] Dialog Statements End Dialog
Description	Defines the end of the dialog box template for use with the Dialog statement and function.
See Also	Begin Dialog (on page 348) (statement); CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DlgProc (on page 440) (function); DropListBox

	<p>(on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PictureButton (on page 659) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement).</p>
Note	Within user dialog boxes, the default font is 8-point MS Sans Serif.

Environ, Environ\$ (functions)

Syn- tax	Environ [\$](variable\$ VariableNumber)
De- scrip- tion	Returns the value of the specified environment variable.
Com- ments	<p>Environ\$ returns a String , whereas Environ returns a String variant. If variable\$ is specified, then this function looks for that variable\$ in the environment. If the variable\$ name cannot be found, then a zero-length string is returned. If VariableNumber is specified, then this function looks for the Nth variable within the environment (the first variable being number 1). If there is no such environment variable, then a zero-length string is returned. Otherwise, the entire entry from the environment is returned in the following format:</p> <pre>variable = value</pre>
Exam- ple	<p>This example looks for the DOS Comspec variable and displays the value in a dialog box.</p> <pre>Sub Main() Dim a\$(1) a\$(1) = Environ("SITE_Root") MsgBox "My SIMPLICITY project directory is: " & a\$(1) End Sub</pre>
See Also	Command (on page 375), Command\$ (on page 375) (functions).

EOF (function)

Syn- tax	EOF (filenumber)
-------------	-------------------------

De- scrip- tion	Returns True if the end-of-file has been reached for the given file; returns False otherwise.
Com- ments	The filenumber parameter is an Integer used by the Basic Control Engine to refer to the open file—the number passed to the Open statement. With sequential files, EOF returns True when the end of the file has been reached (i.e., the next file read command will result in a runtime error). With Random or Binary files, EOF returns True after an attempt has been made to read beyond the end of the file. Thus, EOF will only return True when Get was unable to read the entire record.
Exam- ple	<p>This example opens the autoexec.bat file and reads lines from the file until the end-of-file is reached.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() file\$ = "c:\autoexec.bat" Open file\$ For Input As #1 Do While Not EOF(1) Line Input #1,newline Loop Close MsgBox "The last line of '" & file\$ "' is:" & crlf & crlf & newline End Sub </pre>
See Also	Open (on page 642) (statement); LOF (on page 597) (function).

Eqv (operator)

Syn- tax	Expression1 Eqv expression2
De- scrip- tion	Performs a logical or binary equivalence on two expressions.
Com- ments	If both expressions are either Boolean , Boolean variants, or Null variants, then a logical equivalence is performed as follows:

	If the first expression is	and the second expression is	then the result is			
	TRUE	TRUE	TRUE			
	TRUE	FALSE	FALSE			
	FALSE	TRUE	FALSE			
	FALSE	FALSE	TRUE			
If either expression is Null , then Null is returned.						
Binary Equivalence If the two expressions are Integer , then a binary equivalence is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary equivalence is then performed, returning a Long result. Binary equivalence forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:						
	1	Eqv	1	=	1	Example
	0	Eqv	1	=	0	5 01101001
	1	Eqv	0	=	0	6 10101010
	0	Eqv	0	=	1	Eqv 00101000
Example	<p>This example assigns False to A, performs some equivalent operations, and displays a dialog box with the result. Since A is equivalent to False, and False is equivalent to 0, and by definition, A = 0, then the dialog box will display "A is False."</p> <pre> Sub Main() a = False If ((a Eqv False) And (False Eqv 0) And (a = 0)) Then MsgBox "a is False." Else MsgBox "a is True." End If End Sub </pre>					
See Also	Operator Precedence (on page 648) (topic); Or (on page 654) (operator); Xor (on page 795) (operator); Imp (on page 557) (operator); And (on page 309) (operator).					

Erase (statement)

Syntax	Erase array1 [,array2]...																								
Description	Erases the elements of the specified arrays.																								
Comments	For dynamic arrays, the elements are erased, and the array is redimensioned to have no dimensions (and therefore no elements). For fixed arrays, only the elements are erased; the array dimensions are not changed.																								
	After a dynamic array is erased, the array will contain no elements and no dimensions. Thus, before the array can be used by your program, the dimensions must be reestablished using the Redim statement. Up to 32 parameters can be specified with the Erase statement.																								
	The meaning of erasing an array element depends on the type of the element being erased:																								
	<table border="1"> <thead> <tr> <th>Element Type</th> <th>What Erase Does to That Element</th> </tr> </thead> <tbody> <tr> <td>Integer</td> <td>Sets the element to 0.</td> </tr> <tr> <td>Boolean</td> <td>Sets the element to FALSE.</td> </tr> <tr> <td>Long</td> <td>Sets the element to 0.</td> </tr> <tr> <td>Double</td> <td>Sets the element to 0.0.</td> </tr> <tr> <td>Date</td> <td>Sets the element to December 30, 1899.</td> </tr> <tr> <td>Single</td> <td>Sets the element to 0.0.</td> </tr> <tr> <td>String (variable-length)</td> <td>Frees the string, then sets the element to a zero-length string.</td> </tr> <tr> <td>String (fixed-length)</td> <td>Sets every character of each element to zero (Chr\$(0)).</td> </tr> <tr> <td>Object</td> <td>Decrements the reference count and sets the element to Nothing .</td> </tr> <tr> <td>Variant</td> <td>Sets the element to Empty .</td> </tr> <tr> <td>User-defined type</td> <td>Sets each structure element as a separate variable.</td> </tr> </tbody> </table>	Element Type	What Erase Does to That Element	Integer	Sets the element to 0.	Boolean	Sets the element to FALSE.	Long	Sets the element to 0.	Double	Sets the element to 0.0.	Date	Sets the element to December 30, 1899.	Single	Sets the element to 0.0.	String (variable-length)	Frees the string, then sets the element to a zero-length string.	String (fixed-length)	Sets every character of each element to zero (Chr\$(0)).	Object	Decrements the reference count and sets the element to Nothing .	Variant	Sets the element to Empty .	User-defined type	Sets each structure element as a separate variable.
Element Type	What Erase Does to That Element																								
Integer	Sets the element to 0.																								
Boolean	Sets the element to FALSE.																								
Long	Sets the element to 0.																								
Double	Sets the element to 0.0.																								
Date	Sets the element to December 30, 1899.																								
Single	Sets the element to 0.0.																								
String (variable-length)	Frees the string, then sets the element to a zero-length string.																								
String (fixed-length)	Sets every character of each element to zero (Chr\$(0)).																								
Object	Decrements the reference count and sets the element to Nothing .																								
Variant	Sets the element to Empty .																								
User-defined type	Sets each structure element as a separate variable.																								
Example	<p>This example fills an array with a list of available disk drives, displays the list, erases the array and then redisplay the list.</p> <pre> Sub Main() Dim a\$(10) 'Declare an array. DiskDrives a 'Fill element 1 with a list of available disk drives. r = SelectBox("Array Before Erase",,a) </pre>																								

	<pre>Erase a\$ 'Erase all elements in the array. r = SelectBox("Array After Erase", ,a) End Sub</pre>
See Also	Redim (on page 688) (statement); Arrays (on page 329) (topic).

Erl (function)

Syn-tax	Erl[0]
Description	Returns the line number of the most recent error.
Comments	The first line of the script is 1, the second line is 2, and so on. The internal value of Erl is reset to 0 with any of the following statements: Resume , Exit Sub , Exit Function . Thus, if you want to use this value outside an error handler, you must assign it to a variable.
Example	<p>This example generates an error and then determines the line on which the error occurred.</p> <pre>Sub Main() Dim i As Integer On Error Goto Trap1 i = 32767 'Generate an error--overflow. i = i + 1 Exit Sub Trap1: MsgBox "Error on line: " & Erl Exit Sub 'Reset the error handler. End Sub</pre>
See Also	Err (on page 492) (function); Error, Error\$ (on page 493) (functions); Error Handling (on page 495) (topic).

Err (function)

This function is obsolete.

Refer to [Err.Number \(property\) \(on page 501\)](#) .

Err (statement)

Syn- tax	Err = value
De- scrip- tion	Sets the value returned by the Err function to a specific Integer value.
Com- ments	Only positive values less than or equal to 32767 can be used. Setting value to -1 has the side effect of resetting the error state. This allows you to perform error trapping within an error handler. The ability to reset the error handler while within an error trap is not standard Basic. Normally, the error handler is reset only with the Resume , Exit Sub , or Exit Function statement.
Exam- ple	<p>This example forces error 10, with a subsequent transfer to the TestError label. TestError tests the error and, if not error 55, resets Err to 999 (user-defined error) and returns to the Main subroutine.</p> <pre> Sub Main() On Error Goto TestError Error 10 MsgBox "The returned error is: " & Err() & " - " & Error\$ & "" Exit Sub TestError: If Err = 55 Then 'File already open. MsgBox "Cannot copy an open file. Close it and try again." Else MsgBox "Error '" & Err & "' has occurred." Err = 999 End If Resume Next End Sub </pre>
See Also	Error (on page 494) (statement); Error Handling (on page 495) (topic).

Error, Error\$ (functions)

Syn- tax	Error[\$]([errornumber])
-------------	---------------------------------

De- scrip- tion	Returns a String containing the text corresponding to the given error number or the most recent error.
Com- ments	Error\$ returns a String , whereas Error returns a String variant. The errornumber parameter is an Integer containing the number of the error message to retrieve. If this parameter is omitted, then the function returns the text corresponding to the most recent runtime error. If no runtime error has occurred, then a zero-length string is returned. If the Error statement was used to generate a user-defined runtime error, then this function will return a zero-length string ("").
Exam- ple	<p>This example forces error 10, with a subsequent transfer to the TestError label. TestError tests the error and, if not error 55, resets Err to 999 (user-defined error) and returns to the Main subroutine.</p> <pre> Sub Main() On Error Goto TestError Error 10 MsgBox "The returned error is: '" & Err & "' - '" & Error & "'" Exit Sub TestError: If Err = 55 Then 'File already open. MsgBox "Cannot copy an open file. Close it and try again." Else MsgBox "Error '" & Err & "' has occurred." Err = 999 End If Resume Next End Sub </pre>
See Also	Erl (on page 492) (function); Err (on page 492) (function); Error Handling (on page 495) (topic).

Error (statement)

Syn- tax	Error errornumber
De- scrip- tion	Simulates the occurrence of the given runtime error.

Com- ments	The errornumber parameter is any Integer containing either a built-in error number or a user-defined error number. The Err function can be used within the error trap handler to determine the value of the error.
Exam- ple	<p>This example forces error 10, with a subsequent transfer to the TestError label. TestError tests the error and, if not error 55, resets Err to 999 (user-defined error) and returns to the Main sub-routine.</p> <pre data-bbox="292 504 1421 1134"> Sub Main() On Error Goto TestError Error 10 MsgBox "The returned error is: " & Err() & " - " & Error\$ & "" Exit Sub TestError: If Err = 55 Then 'File already open. MsgBox "Cannot copy an open file. Close it and try again." Else MsgBox "Error " & Err & " has occurred." Err = 999 End If Resume Next End Sub </pre>
See Also	Err (on page 493) (statement); Error Handling (on page 495) (topic).

Error Handling (topic)

1. **Visual Basic-compatible errors:** These errors, numbered between 0 and 799, are numbered and named according to the errors supported by Visual Basic.
2. **Basic Control Engine script errors:** These errors, numbered from 800 to 999, are unique to the Basic Control Engine..
3. **User-defined errors:** These errors, equal to or greater than 1,000, are available for use by extensions or by the script itself.

Err.Clear (method)

Syntax	Err.Clear	
Description	Clears the properties of the Err object.	
Comments	After this method has been called, the properties of the Err object will have the following values:	
	Property	Value
	Err.Description	""
	Err.HelpContext	0
	Err.HelpFile	""
	Err.LastDLLError	0
	Err.Number	0
	Err.Source	""
	<p>The properties of the <code>Err</code> object are automatically reset when any of the following statements are executed:</p> <pre> Resume Exit Function On Error Exit Sub </pre>	
Example	<pre> ' The following script gets input from the user using error ' checking. Sub Main() Dim x As Integer On Error Resume Next x = InputBox("Type in a number") If Err.Number <> 0 Then Err.Clear x = 0 End If MsgBox x </pre>	

	End Sub
See Also	Error Handling (on page 495) (topic), Err.Description (on page 497) (property), Err.HelpContext (on page 498) (property), Err.HelpFile (on page 499) (property), Err.LastDLLError (on page 500) (property), Err.Number (on page 501) (property), Err.Source (on page 504) (property)

Err.Description (property)

Syntax	<code>Err.Description [= stringexpression]</code>
Description	Sets or retrieves the description of the error.
Comments	For errors generated by BasicScript, the <code>Err.Description</code> property is automatically set. For user-defined errors, you should set this property to be a description of your error. If you set the <code>Err.Number</code> property to one of BasicScript's internal error numbers and you don't set the <code>Err.Description</code> property, then the <code>Err.Description</code> property is automatically set when the error is generated (i.e., with <code>Err.Raise</code>).
Example	<pre> 'The following script gets input from the user using error 'checking. When an error occurs, the Err.Description property 'is displayed to the user and execution continues with a default 'value. Sub Main() Dim x As Integer On Error Resume Next x = InputBox("Type in a number") If Err.Number <> 0 Then MsgBox "The following error occurred: " & Err.Description End If MsgBox x End Sub </pre>

See Also	Error Handling (on page 495) (topic), Err.Clear (on page 495) (method), Err.HelpContext (on page 498) (property), Err.HelpFile (on page 499) (property), Err.LastDLLError (on page 500) (property), Err.Number (on page 501) (property), Err.Source (on page 504) (property)
-------------	--

Err.HelpContext (property)

Syntax	<code>Err.HelpContext [= contextid]</code>
Description	Sets or retrieves the help context ID that identifies the help topic for information on the error.
Comments	The <code>Err.HelpContext</code> property, together with the <code>Err.HelpFile</code> property, contain sufficient information to display help for the error. When BasicScript generates an error, the <code>Err.HelpContext</code> property is set to 0 and the <code>Err.HelpFile</code> property is set to ""; the value of the <code>Err.Number</code> property is sufficient for displaying help in this case. The exception is with errors generated by an OLE automation server; both the <code>Err.HelpFile</code> and <code>Err.HelpContext</code> properties are set by the server to values appropriate for the generated error. When generating your own user-defined errors, you should set the <code>Err.HelpContext</code> property and the <code>Err.HelpFile</code> property appropriately for your error. If these are not set, then BasicScript displays its own help at an appropriate place.
Example	<pre> ' This example defines a replacement for InputBox that deals 'specifically with Integer values. If an error occurs, the 'function generates a user-defined error that can be trapped 'by the caller. Function InputInteger(Prompt,Optional Title,Optional Def) On Error Resume Next Dim x As Integer x = InputBox(Prompt,Title,Def) If Err.Number Then Err.HelpFile = "AZ.HLP" Err.HelpContext = 2 Err.Description = "Integer value expected" InputInteger = Null Err.Raise 3000 End If InputInteger = x </pre>

	<pre> End Function Sub Main Dim x As Integer Do On Error Resume Next x = InputInteger("Enter a number:") If Err.Number = 3000 Then MsgBox "Invalid number, press ""F1"" to invoke help" _ ,,,Err.HelpFile,Err.HelpContext End If Loop Until Err.Number <> 3000 End Sub </pre>
See Also	Error Handling (on page 495) (topic), Err.Clear (on page 495) (method), Err.Description (on page 497) (property), Err.HelpFile (on page 499) (property), Err.LastDLLError (on page 500) (property), Err.Number (on page 501) (property), Err.Source (on page 504) (property)

Err.HelpFile (property)

Syntax	Err.HelpFile [= filename]
Description	Sets or retrieves the name of the help file associated with the error.
Comments	<p>The <code>Err.HelpFile</code> property, together with the <code>Err.HelpContents</code> property, contain sufficient information to display help for the error. When BasicScript generates an error, the <code>Err.HelpContents</code> property is set to 0 and the <code>Err.HelpFile</code> property is set to ""; the value of the <code>Err.Number</code> property is sufficient for displaying help in this case. The exception is with errors generated by an OLE automation server; both the <code>Err.HelpFile</code> and <code>Err.HelpContext</code> properties are set by the server to values appropriate for the generated error. When generating your own user-define errors, you should set the <code>Err.HelpContext</code> property and the <code>Err.HelpFile</code> property appropriately for your error. If these are not set, then BasicScript displays its own help at an appropriate place.</p>
Example	<pre> ' This example defines a replacement for InputBox that deals 'specifically with Integer values. If an error occurs, the </pre>

```

'function generates a user-defined error that can be trapped
'by the caller.
Function InputInteger(Prompt,Optional Title,Optional Def)
On Error Resume Next
Dim x As Integer
x = InputBox(Prompt,Title,Def)
If Err.Number Then
Err.HelpFile = "AZ.HLP"
Err.HelpContext = 2
Err.Description = "Integer value expected"
InputInteger = Null
Err.Raise 3000
End If
InputInteger = x
End Function
Sub Main
Dim x As Integer
Do
On Error Resume Next
x = InputInteger("Enter a number:")
If Err.Number = 3000 Then
MsgBox "Invalid number, press ""F1"" to invoke help" _
,,, Err.HelpFile,Err.HelpContext
End If
Loop Until Err.Number <> 3000
End Sub

```

See [Error Handling \(on page 495\)](#) (topic) [Err.Clear \(on page 495\)](#) (method), [Err.HelpContext \(on page 498\)](#) (property), [Err.Description \(on page 497\)](#)(property), [Err.LastDLLError \(on page 500\)](#) (property), [Err.Number \(on page 501\)](#) (property), [Err.Source \(on page 504\)](#) (property)

Note The Err.HelpFile property can be set to any valid Windows help file (i.e., a file with a .HLP extension compatible with the WINHELP help engine).

Err.LastDLLError (property)

Syntax	<code>Err.LastDLLError</code>
Description	Returns the last error generated by an external call, i.e. a call to a routine declared with the <code>Declare</code> statement that resides in an external module.
Comments	The <code>Err.LastDLLError</code> property is automatically set when calling a routine defined in an external module. If no error occurs within the external call this property will automatically be set to 0.
Example	<pre>'The following script calls the GetCurrentDirectoryA. If an 'error occurs, this Win32 function sets the Err.LastDLLError 'property which can be checked for. Declare Sub GetCurrentDirectoryA Lib "kernel32" (ByVal DestLen _ As Integer,ByVal lpDest As String) Sub Main() Dim dest As String * 256 Err.Clear GetCurrentDirectoryA len(dest),dest If Err.LastDLLError <> 0 Then MsgBox "Error " & Err.LastDLLError & " occurred." Else MsgBox "Current directory is " & dest End If End Sub</pre>
See Also	Error Handling (on page 495) (topic), Err.Clear (on page 495) (method), Err.HelpContext (on page 498) (property), Err.Description (on page 497) (property), Err.HelpFile (on page 499) (property), Err.Number (on page 501) (property), Err.Source (on page 504) (property)
Note	This property is set by DLL routines that set the last error using the Win32 function <code>SetLastError()</code> . BasicScript uses the Win32 function <code>GetLastError()</code> to retrieve the value of this property. The value 0 is returned when calling DLL routines that do not set an error.

Err.Number (property)

Syntax	<code>Err.Number [= errornumber]</code>
--------	---

De- scrip- tion	Returns or sets the number of the error.
Com- ments	<p>The <code>Err.Number</code> property is set automatically when an error occurs. This property can be used within an error trap to determine which error occurred. You can set the <code>Err.Number</code> property to any Long value. The <code>Number</code> property is the default property of the <code>Err</code> object. This allows you to use older style syntax such as those shown below: Err = 6 If Err = 6 Then MsgBox "Overflow"</p> <p>The <code>Err</code> function can only be used while within an error trap. The internal value of the <code>Err.Number</code> property is reset to 0 with any of the following statements: <code>Resume</code>, <code>Exit Sub</code>, <code>Exit Function</code>. Thus, if you want to use this value outside an error handler, you must assign it to a variable. Setting <code>Err.Number</code> to -1 has the side effect of resetting the error state. This allows you to perform error trapping within an error handler. The ability to reset the error handler while within an error trap is not standard Basic. Normally, the error handler is reset only with the <code>Resume</code>, <code>Exit Sub</code>, <code>Exit Function</code>, <code>End Function</code>, or <code>End Sub</code> statements.</p>
Exam- ple	<pre> 'This example forces error 10, with a subsequent transfer to 'the TestError label. TestError tests the error and, if not 'error 55, resets Err to 999 (user-defined error) and returns 'to the Main subroutine. Sub Main() On Error Goto TestError Error 10 MsgBox "The returned error is: '" & Err() & "' - '" & _ Error\$ & "'" Exit Sub TestError: If Err = 55 Then 'File already open. MsgBox "Cannot copy an open file. Close it and try again." Else MsgBox "Error '" & Err & "' has occurred!" Err = 999 End If Resume Next End Sub </pre>

See Also	Error Handling (on page 495) (topic)
----------	--

Err.Raise (method)

Syn-tax	<code>Err.Raise</code> number [, [source] [, [description] [, [helpfile] [, helpcontext]]]]	
De-scrip-tion	Generates a runtime error, setting the specified properties of the <code>Err</code> object.	
Com-ments	The <code>Err.Raise</code> method has the following named parameters:	
	Para-meter	Description
	num-ber	A Long value indicating the error number to be generated. This parameter is required. Errors predefined by BasicScript are in the range between 0 and 1000.
	source	An optional String expression specifying the source of the error—i.e., the object or module that generated the error. If omitted, then BasicScript uses the name of the currently executing script.
	de-scrip-tion	An optional String expression describing the error. If omitted and number maps to a predefined BasicScript error number, then the corresponding predefined description is used. Otherwise, the error "Application-defined or object-define error" is used.
	help-file	An optional String expression specifying the name of the help file containing context-sensitive help for this error. If omitted and number maps to a predefined BasicScript error number, then the default help file is assumed.
	help-con-text	An optional Long value specifying the topic within helpfile containing context-sensitive help for this error. If some arguments are omitted, then the current property values of the <code>Err</code> object are used. This method can be used in place of the Error statement for generating errors. Using the <code>Err.Raise</code> method gives you the opportunity to set the desired properties of the <code>Err</code> object in one statement.
Exam-ple	<pre>'The following example uses the Err.Raise method to generate 'a user-defined error.</pre>	

	<pre> Sub Main() Dim x As Variant On Error Goto TRAP x = InputBox("Enter a number:") If Not IsNumber(x) Then Err.Raise 3000,,"Invalid number specified","WIDGET.HLP",30 End If MsgBox x Exit Sub TRAP: MsgBox Err.Description End Sub </pre>
See Also	<p>Error (on page 494) (statement), Error Handling (on page 495) (topic), Err.Clear (on page 495) (method), Err.HelpContext (on page 498) (property), Err.Description (on page 497) (property), Err.HelpFile (on page 499) (property), Err.Number (on page 501) (property), Err.Source (on page 504) (property)</p>

Err.Source (property)

Syntax	<code>Err.Source [= stringexpression]</code>
Description	Sets or retrieves the source of a runtime error.
Comments	<p>For OLE automation errors generated by the OLE server, the <code>Err.Source</code> property is set to the name of the object that generated the error. For all other errors generated by BasicScript, the <code>Err.Source</code> property is automatically set to be the name of the script that generated the error. For user-defined errors, the <code>Err.Source</code> property can be set to any valid String expression indicating the source of the error. If the <code>Err.Source</code> property is not explicitly set for user-defined errors, the BasicScript sets the value to be the name of the script in which the error was generated.</p>
Example	<pre> 'The following script generates an error, setting the source 'to the specific location where the error was generated. Function InputInteger(Prompt,Optional Title,Optional Def) </pre>

	<pre> On Error Resume Next Dim x As Integer x = InputBox(Prompt,Title,Def) If Err.Number Then Err.Source = "InputInteger" Err.Description = "Integer value expected" InputInteger = Null Err.Raise 3000 End If InputInteger = x End Function Sub Main On Error Resume Next x = InputInteger("Enter a number:") If Err.Number Then MsgBox Err.Source & ":" & Err.Description End Sub </pre>
See Also	Error Handling (on page 495) (topic), Err.Clear (on page 495) (method), Err.HelpContext (on page 498) (property), Err.Description (on page 497) (property), Err.HelpFile (on page 499) (property), Err.Number (on page 501) (property), Err.LastDLLError (on page 500) (property)

Exit Do (statement)

Syntax	Exit Do
Description	Causes execution to continue on the statement following the Loop clause.
Comments	This statement can only appear within a Do...Loop statement.
Example	<p>This example will load an array with directory entries unless there are more than ten entries-in which case, the Exit Do terminates the loop.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a\$(5) Do </pre>

	<pre> i% = i% + 1 If i% = 1 Then a(i%) = Dir("**") Else a(i%) = Dir End If If i% >= 5 Then Exit Do Loop While (a(i%) <> "") If i% = 5 Then MsgBox i% & " directory entries processed!" Else MsgBox "Less than " & i% & " entries processed!" End If End Sub </pre>
See Also	<p>Stop (on page 737) (statement); Exit For (on page 506) (statement); Exit Function (on page 507) (statement); Exit Sub (on page 507) (statement); End (on page 487) (statement); Do...Loop (on page 454) (statement).</p>

Exit For (statement)

Syntax	Exit For
Description	Causes execution to exit the innermost For loop, continuing execution on the line following the Next statement.
Comments	This statement can only appear within a For...Next block.
Example	<p>This example enters a large user-defined cycle, performs a calculation and exits the For...Next loop when the result exceeds a certain value.</p> <pre> Const critical_level = 500 Sub Main() num = InputBox("Please enter the number of cycles","Cycles") For i = 1 To Val(num) newpressure = i * 2 If newpressure >= critical_level Then Exit For Next i </pre>

	<pre>MsgBox "The valve pressure is: " & newpressure End Sub</pre>
See Also	Stop (on page 737) (statement); Exit Do (on page 505) (statement); Exit Function (on page 507) (statement); Exit Sub (on page 507) (statement); End (on page 487) (statement); Do...Loop (on page 454) (statement).

Exit Function (statement)

Syntax	Exit Function
Description	Causes execution to exit the current function, continuing execution on the statement following the call to this function.
Comments	This statement can only appear within a function.
Example	<p>This function displays a message and then terminates with Exit Function.</p> <pre>Function Test_Exit() As Integer MsgBox "Testing function exit, returning to Main()." Test_Exit = 0 Exit Function MsgBox "This line should never execute." End Function Sub Main() a% = Test_Exit() MsgBox "This is the last line of Main()." End Sub</pre>
See Also	Stop (on page 737) (statement); Exit For (on page 506) (statement); Exit Do (on page 505) (statement); Exit Sub (on page 507) (statement); End (on page 487) (statement); Do...Loop (on page 454) (statement).

Exit Sub (statement)

Syntax	Exit Sub
--------	-----------------

De- scrip- tion	Causes execution to exit the current subroutine, continuing execution on the statement following the call to this subroutine.
Com- ments	This statement can appear anywhere within a subroutine. It cannot appear within a function.
Exam- ple	<p>This example displays a dialog box and then exits. The last line should never execute because of the Exit Sub statement.</p> <pre>Sub Main() MsgBox "Terminating Main()." Exit Sub MsgBox "Still here in Main()." End Sub</pre>
See Al- so	Stop (on page 737) (statement); Exit For (on page 506) (statement); Exit Function (on page 507) (statement); Exit Do (on page 505) (statement); End (on page 487) (statement); Do...Loop (on page 454) (statement).

Exp (function)

Syntax	Exp (value)
Descrip- tion	Returns the value of e raised to the power of value.
Com- ments	<p>The value parameter is a Double within the following range:</p> <pre>0 <= value <= 709.782712893.</pre> <p>A runtime error is generated if value is out of the range specified above. The value of e is 2.71828.</p>
Example	<p>This example assigns a to e raised to the 12.4 power and displays it in a dialog box.</p> <pre>Sub Main() a# = Exp(12.4) MsgBox "e to the 12.4 power is: " & a# End Sub</pre>
See Also	Log (on page 598) (function).

Expression Evaluation (topic)

Basic Control Engine scripts allows expressions to involve data of different types. When this occurs, the two arguments are converted to be of the same type by promoting the less precise operand to the same type as the more precise operand. For example, the Basic Control Engine will promote the value of **i%** to a **Double** in the following expression:

```
result# = i% * d#
```

In some cases, the data type to which each operand is promoted is different than that of the most precise operand. This is dependent on the operator and the data types of the two operands and is noted in the description of each operator.

If an operation is performed between a numeric expression and a **String** expression, then the **String** expression is usually converted to be of the same type as the numeric expression. For example, the following expression converts the **String** expression to an **Integer** before performing the multiplication:

```
result = 10 * "2"      'Result is equal to 20.
```

There are exceptions to this rule as noted in the description of the individual operators.

Type Coercion

The Basic Control Engine performs numeric type conversion automatically. Automatic conversions sometimes result in overflow errors, as shown in the following example:

```
d# = 45354
i% = d#
```

In this example, an overflow error is generated because the value contained in **d#** is larger than the maximum size of an **Integer**.

Rounding

When floating-point values (**Single** or **Double**) are converted to integer values (**Integer** or **Long**), the fractional part of the floating-point number is lost, rounding to the nearest integer value. The Basic Control Engine uses Baker's rounding:

- If the fractional part is larger than .5, the number is rounded up.
- If the fractional part is smaller than .5, the number is rounded down.
- If the fractional part is equal to .5, then the number is rounded up if it is odd and down if it is even.

The following table shows sample values before and after rounding:

Before Rounding	After Rounding to Whole Number
2.1	2
4.6	5
2.5	2
3.5	4

Default Properties

When an OLE object variable or an **Object** variant is used with numerical operators such as addition or subtraction, then the default property of that object is automatically retrieved. For example, consider the following:

```
Dim Excel As Object
Set Excel = GetObject(,"Excel.Application")
MsgBox "This application is " & Excel
```

The above example displays **This application is Microsoft Excel** in a dialog box. When the variable **Excel** is used within the expression, the default property is automatically retrieved, which, in this case, is the string **Microsoft Excel**. Considering that the default property of the **Excel** object is **.Value**, then the following two statements are equivalent:

```
MsgBox "This application is " & Excel
MsgBox "This application is " & Excel.Value
```

F

F

False (constant)
FileAttr (function)
FileCopy (statement)
FileDateTime (function)
FileDirs (statement)

FileExists (function)
FileLen (function)
FileList (statement)
FileParse\$ (function)
Fix (function)
For Each...Next (statement)
For...Next (statement)
Format, Format\$ (function)
FreeFile (function)
Function...End Function (statement)
Fv Function...End Function (statement)

False (constant)

De- scrip- tion	Boolean constant whose value is False.
Com- ments	Used in conditionals and Boolean expressions.
Exam- ple	<p>This example assigns False to a, performs some equivalent operations, and displays a dialog box with the result. Since a is equivalent to False, and False is equivalent to 0, and by definition, a = 0, then the dialog box will display "a is False."</p> <pre> Sub Main() a = False If ((a = False) And (False Eqv 0) And (a = 0)) Then MsgBox "a is False." Else MsgBox "a is True." End If End Sub </pre>

See	True (on page 760) (constant); Constants (topic); Boolean (on page 351) (data type).
Also	

FileAttr (function)

Syn- tax	FileAttr (filename, attribute)	
De- scrip- tion	Returns an Integer specifying the file mode (if attribute is 1) or the operating system file handle (if attribute is 2).	
Com- ments	The FileAttr function takes the following parameters:	
	Parameter	Description
	Filename	Integer value used by Basic Control Engine to refer to the open file; the number passed to the Open statement.
	Attribute	Integer specifying the type of value to be returned. If attribute is 1, then one of the following values is returned:
		1 Input
		2 Output
		4 Random
		8 Append
		32 Binary
		If attribute is 2, then the operating system file handle is returned. On most systems, this is a special Integer value identifying the file.
Exam- ple	<p>This example opens a file for input, reads the file attributes, and determines the file mode for which it was opened. The result is displayed in a dialog box.</p> <pre> Sub Main() Open "c:\autoexec.bat" For Input As #1 a% = FileAttr(1,1) Select Case a% Case 1 MsgBox "Opened for input." </pre>	

```

Case 2

    MsgBox "Opened for output."

Case 4

    MsgBox "Opened for random."

Case 8

    MsgBox "Opened for append."

Case 32

    MsgBox "Opened for binary."

Case Else

    MsgBox "Unknown file mode."

End Select

a% = FileAttr(1,2)

MsgBox "File handle is: " & a%

Close

End Sub

```

See [FileLen \(on page 517\)](#) (function); [GetAttr \(on page 538\)](#) (function); [FileExists \(on page 516\)](#) (function); [Open \(on page 642\)](#) (statement); [SetAttr \(on page 715\)](#) (statement).

FileCopy (statement)

Syn- tax	FileCopy source\$, destination\$	
De- scrip- tion	Copies a source\$ file to a destination\$ file.	
Com- ments	The FileCopy function takes the following parameters:	
	Para- meter	Description
	source \$	String containing the name of a single file to copy. The source\$ parameter cannot contain wildcards (? or *) but may contain path information.
	des- tina- tion\$	String containing a single, unique destination file, which may contain a drive and path specification.

	The file will be copied and renamed if the source\$ and destination\$ filenames are not the same. Some platforms do not support drive letters and may not support dots to indicate current and parent directories.
Example	<p>This example copies the autoexec.bat file to "autoexec.sav", then opens the copied file and tries to copy it again--which generates an error.</p> <pre> Sub Main() On Error Goto ErrHandler FileCopy "c:\autoexec.bat", "c:\autoexec.sav" Open "c:\autoexec.sav" For Input As # 1 FileCopy "c:\autoexec.sav", "c:\autoexec.sv2" Close Exit Sub ErrHandler: If Err = 55 Then 'File already open. MsgBox "Cannot copy an open file. Close it and try again." Else MsgBox "An unspecified file copy error has occurred." End If Resume Next End Sub </pre>
See Also	Kill (on page 579) (statement); Name (on page 619) (statement).

FileDateTime (function)

Syn- tax	FileDateTime (filename\$)
De- scrip- tion	Returns a Date variant representing the date and time of the last modification of a file.
Com- ments	This function retrieves the date and time of the last modification of the file specified by filename\$ (wildcards are not allowed). A runtime error results if the file does not exist. The value returned can be used with the date/time functions (i.e., Year , Month , Day , Weekday , Minute , Second , Hour) to extract the individual elements.

Example	<p>This example gets the file date/time of the autoexec.bat file and displays it in a dialog box.</p> <pre> Sub Main() If FileExists("c:\autoexec.bat") Then a# = FileDateTime("c:\autoexec.bat") MsgBox "The date/time information for the file is: " & Year(a#) & "-" & Month(a#) & "-" & Day(a#) Else MsgBox "The file does not exist." End If End Sub </pre>
See Also	FileLen (on page 517) (function); GetAttr (on page 538) (function); FileAttr (on page 512) (function); FileExists (on page 516) (function).
Notes:	<p>The Win32 operating system stores the file creation date, last modification date, and the date the file was last written to. The FileDateTime function only returns the last modification date.</p>

FileDirs (statement)

Syntax	FileDirs array() [,dirspec\$]	
Description	Fills a String or Variant array with directory names from disk.	
Comments	The FileDirs statement takes the following parameters:	
	Parameter	Description
	Array()	Either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed. If array() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the <code>LBound</code> , <code>UBound</code> , and <code>ArrayDims</code> functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.

	Dirspec \$	String containing the file search mask, such as:
		<code>t*. c:*</code>
		If this parameter is omitted, then <code>*</code> is used, which fills the array with all the subdirectory names within the current directory.
Example		<p>This example fills an array with directory entries and displays the first one.</p> <pre>Sub Main() Dim a\$() FileDirs a\$, "c:*" MsgBox "The first directory is: " & a\$(0) End Sub</pre>
See Also		FileList (on page 517) (statement); Dir, Dir\$ (on page 427) (functions); CurDir, CurDir\$ (on page 387) (functions); ChDir (on page 359) (statement).

FileExists (function)

Syntax	FileExists (filename\$)
Description	Returns True if filename\$ exists; returns False otherwise.
Comments	This function determines whether a given filename\$ is valid. This function will return False if filename\$ specifies a subdirectory.
Example	<p>This example checks to see whether there is an autoexec.bat file in the root directory of the C drive, then displays either its creation date and time or the fact that it does not exist.</p> <pre>Sub Main() If FileExists("c:\autoexec.bat") Then MsgBox "This file exists!" Else MsgBox "File does not exist." End If End Sub</pre>

See	FileLen (on page 517) (function); GetAttr (on page 538) (function); FileAttr (on page 512)
Also	(function); FileParse\$ (on page 520) (function).

FileLen (function)

Syn-tax	FileLen (filename\$)
De-scrip-tion	Returns a Long representing the length of filename\$ in bytes.
Com-ments	This function is used in place of the LOF function to retrieve the length of a file without first opening the file. A runtime error results if the file does not exist.
Exam-ple	<p>This example checks to see whether there is a c:\autoexec.bat file and, if there is, displays the length of the file.</p> <pre> Sub Main() file\$ = "c:\autoexec.bat" If FileExists(file\$) And FileLen(file\$) <> 0 Then b% = FileLen(file\$) MsgBox "" & file\$ & " is " & b% & " bytes." Else MsgBox "" & file\$ & " does not exist." End If End Sub </pre>
See	GetAttr (on page 538) (function); FileAttr (on page 512) (function); FileParse\$ (on page 520) (function); FileExists (on page 516) (function); Loc (on page 594) (function).
Also	

FileList (statement)

Syn-tax	FileList array() [[filespec\$] [[include_attr] [exclude_attr]]]
De-scrip-tion	Fills a String or Variant array with filenames from disk.

Com-ments	The FileList function takes the following parameters:		
	Parame-ter	Description	
	Array()	Either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed. If array() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the LBound , UBound , and ArrayDims functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.	
	Filespec\$	String specifying which filenames are to be included in the list. The filespec\$ parameter can include wildcards, such as * and ?. If this parameter is omitted, then * is used.	
	Include_-attr	Integer specifying attributes of files you want included in the list. It can be any combination of the attributes listed below. If this parameter is omitted, then the value 97 is used (ebReadOnly Or ebArchive Or ebNone).	
	Exclude_-attr	Integer specifying attributes of files you want excluded from the list. It can be any combination of the attributes listed below. If this parameter is omitted, then the value 18 is used (ebHidden Or ebDirectory). In other words, hidden files and subdirectories are excluded from the list.	
	Wildcards The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple *'s and ?'s can appear within the expression to form complete searching patterns. The following table shows some examples:		
	This Pat-tern	Matches These Files	Doesn't Match These Files
	S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT

C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
C*T	CAT CAP.TXT	CAT.DOC
C?T	CAT CUT	CAT.TXT CAPIT CT
*	(All files)	

File Attributes These numbers can be any combination of the following:

Constant	Value	Includes
EbNormal	0	Read-only, archive, subdir, none
EbRead-Only	1	Read-only files
EbHidden	2	Hidden files
EbSystem	4	System files
EbVolume	8	Volume label
EbDirectory	16	DOS subdirectories
EbArchive	32	Files that have changed since the last backup
EbNone	64	Files with no attributes

Example This example fills an array a with the directory of the current drive for all files that have normal or no attributes and excludes those with system attributes. The dialog box displays four file-names from the array.

```

Const crlf = Chr$(13) + Chr$(10)

Sub Main()

    Dim a$()

    FileList a$,"*.*",(ebNormal + ebNone),ebSystem

    If ArrayDims(a$) > 0 Then

        r = SelectBox("FileList","The files you filtered are:",a$)
    
```


	<pre> Else MsgBox "No files found." End If End Sub </pre>
See Also	FileDirs (on page 515) (statement); Dir, Dir\$ (on page 427) (functions).

FileParse\$ (function)

Syntax	<code>FileParse\$ (filename\$, operation)</code>		
Description	Returns a String containing a portion of filename\$ such as the path, drive, or file extension.		
Comments	<p>The filename\$ parameter can specify any valid filename (it does not have to exist). For example:</p> <pre> ..\test.dat c:\sheets\test.dat test.dat </pre> <p>A runtime error is generated if filename\$ is a zero-length string. The optional operation parameter is an Integer specifying which portion of the filename\$ to extract. It can be any of the following values.</p>		
	Value	Meaning	Example
	0	Full name	c:\sheets\test.dat
	1	Drive	c
	2	Path	c:\sheets
	3	Name	test.dat
	4	Root	test
	5	Extension	dat
	If operation is not specified, then the full name is returned. A runtime error will result if operation is not one of the above values. A runtime error results if filename\$ is empty.		
Example	This example parses the file string <code>c:\temp\autoexec.bat</code> into its component parts and displays them in a dialog box.		

	<pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a\$(5) file\$ = "c:\temp\autoexec.bat" For i = 1 To 5 a\$(i) = FileParse\$(file\$,i) Next i msg1 = "The breakdown of '" & file\$ & "' is:" & crlf & crlf msg1 = msg & a\$(1) & crlf & a\$(2) & crlf & a\$(3) & crlf & a\$(4) & crlf & a\$(5) MsgBox msg1 End Sub </pre>
See Also	FileLen (on page 517) (function); GetAttr (on page 538) (function); FileAttr (on page 512) (function); FileExists (on page 516) (function).
Note	The backslash and forward slash can be used interchangeably. For example, c:\test.dat is the same as c:/test.dat.

Fix (function)

Syntax	Fix (number)
Description	Returns the integer part of number.
Comments	This function returns the integer part of the given value by removing the fractional part. The sign is preserved. The Fix function returns the same type as number, with the following exceptions:
	<ul style="list-style-type: none"> • If number is Empty, then an Integer variant of value 0 is returned. • If number is a String, then a Double variant is returned. • If number contains no valid data, then a Null variant is returned.
Example	<p>This example returns the fixed part of a number and assigns it to b, then displays the result in a dialog box.</p> <pre> Sub Main() a# = -19923.45 b% = Fix(a#) </pre>

	<pre>MsgBox "The fixed portion of -19923.45 is: " & b% End Sub</pre>
See Also	Int (on page 565) (function); CInt (on page 364) (function).

For Each...Next (statement)

Syntax	<code>For Each member in group [statements] [Exit For] [statements] Next [member]</code>	
Description	Repeats a block of statements for each element in a collection or array.	
Comments	The <code>For Each...Next</code> statement takes the following parameters:	
	Parameter	Description
	member	Name of the variable used for each iteration of the loop. If group is an array, then member must be a Variant variable. If group is a collection, then member must be an Object variable, an explicit OLE automation object, or a Variant.
	group	Name of a collection or array.
	statements	Any number of BasicScript statements.
	<p>BasicScript supports iteration through the elements of OLE collections or arrays, unless the arrays contain user-defined types or fixed-length strings. The iteration variable is a copy of the collection or array element in the sense that change to the value of member within the loop has no effect on the collection or array. The <code>For Each...Next</code> statement traverses array elements in the same order the elements are stored in memory. For example, the array elements contained in the array defined by the statement <code>Dim a(1 To 2,3 To 4)</code> are traversed in the following order: (1,3), (1,4), (2,3), (2,4). The order in which the elements are traversed should not be relevant to the correct operation of the script. The <code>For Each...Next</code> statement continues executing until there are no more elements in group or until an <code>Exit For</code> statement is encountered. <code>For Each...Next</code> statements can be nested. In such a case, the <code>Next [member]</code> statement applies to the innermost <code>For Each...Next</code> or <code>For...Next</code> statement. Each member variable of nested <code>For</code></p>	

Each...Next statements must be unique. A `Next` statement appearing by itself (with no member variable) matches the innermost `For Each...Next` or `For...Next` loop.

Exam-
ple

```

        'The following subroutine iterates through the elements
        'of an array using For Each...Next.

Sub Main()

    Dim a(3 To 10) As Single

    Dim i As Variant

    Dim s As String

    For i = 3 To 10

        a(i) = Rnd()

        Next i

For Each i In a

    i = i + 1

Next i

    s = ""

For Each i In a

    If s <> "" Then s = s & ", "

        s = s & i

Next i

MsgBox s

End Sub

        'The following subroutine displays the names of each worksheet
        'in an Excel workbook.

Sub Main()

    Dim Excel As Object

    Dim Sheets As Object

    Set Excel = CreateObject("Excel.Application")

    Excel.Visible = 1

    Excel.Workbooks.Add

    Set Sheets = Excel.Worksheets

For Each a In Sheets

    MsgBox a.Name

Next a

End Sub

```

See Also	Do...Loop (on page 454) (statement), While...Wend (on page 780) (statement), For...Next (on page 524) (statement)
Note	Due to errors in program logic, you can inadvertently create infinite loops in your code. Under Windows and Win32, you can break out of infinite loops using Ctrl+Break.

For...Next (statement)

Syntax	For counter = start To end [Step increment] [statements] [Exit For] [statements] Next [counter [,nextcounter]...]	
Description	Repeats a block of statements a specified number of times, incrementing a loop counter by a given increment each time through the loop.	
Comments	The <code>For</code> statement takes the following parameters:	
	Parameter	Description
	counter	Name of a numeric variable. Variables of the following types can be used: Integer , Long , Single , Double , Variant .
	start	Initial value for counter. The first time through the loop, counter is assigned this value.
	end	Final value for counter. The statements will continue executing until counter is equal to end.
	increment	Amount added to counter each time through the loop. If end is greater than start, then increment must be positive. If end is less than start, then increment must be negative. If increment is not specified, then 1 is assumed. The expression given as increment is evaluated only once. Changing the step during execution of the loop will have no effect.
	statements	Any number of Basic Control Engine statements.
	The For...Next statement continues executing until an Exit For statement is encountered when counter is greater than end. For...Next statements can be nested. In such a case, the Next [counter] statement applies to the innermost For...Next . The Next clause can be optimized for nested next loops by separating each counter with a comma. The ordering of the	

	<p>counters must be consistent with the nesting order (innermost counter appearing before outermost counter). The following example shows two equivalent For statements:</p> <pre> For i = 1 To 10 For i = 1 To 10 For j = 1 To 10 For j = 1 To 10 Next j Next j,i Next i Next i </pre>
	<p>A Next clause appearing by itself (with no counter variable) matches the innermost For loop. The counter variable can be changed within the loop but will have no effect on the number of times the loop will execute.</p>
Example	<pre> Sub Main() 'This example constructs a truth table for the OR statement 'using nested For...Next loops. Msg1 = "Logic table for Or:" & crlf & crlf For x = -1 To 0 For y = -1 To 0 z = x Or y msg1 = msg1 & CBool(x) & " Or " msg1 = msg1 & CBool(y) & " = " msg1 = msg1 & CBool(z) & Basic.Eoln\$ Next y Next x MsgBox msg1 End Sub </pre>
See Also	<p>Do...Loop (on page 454) (statement); While...Wend (on page 780) (statement).</p>
Notes	<p>Due to errors in program logic, you can inadvertently create infinite loops in your code. You can use Ctrl+Break to break out of infinite loops.</p>

Format, Format\$(functions)

Syntax	Format[\$](expression [,Userformat\$])
Description	Returns a String formatted to user specification.

Comments	Format\$ returns a String , whereas Format returns a String variant. The Format\$/Format functions take the following parameters:	
	Parameter	Description
	expression	String or numeric expression to be formatted.
	Userformat\$	Format expression that can be either one of the built-in Basic Control Engine formats or a user-defined format consisting of characters that specify how the expression should be displayed. String, numeric, and date/time formats cannot be mixed in a single Userformat\$ expression.
	If Userformat\$ is omitted and the expression is numeric, then these functions perform the same function as the Str\$ or Str statements, except that they do not preserve a leading space for positive values. If expression is Null , then a zero-length string is returned.	
	Built-In Formats To format numeric expressions, you can specify one of the built-in formats. There are two categories of built-in formats: one deals with numeric expressions and the other with date/time values. The following tables list the built-in numeric and date/time format strings, followed by an explanation of what each does.	
	Numeric Formats	
	Format	Description
	General number	Display the numeric expression as is, with no additional formatting.
	Currency	Displays the numeric expression as currency, with thousands separator if necessary.
	Fixed	Displays at least one digit to the left of the decimal separator and two digits to the right.
	Standard	Displays the numeric expression with thousands separator if necessary. Displays at least one digit to the left of the decimal separator and two digits to the right.
	Percent	Displays the numeric expression multiplied by 100. A percent sign (%) will appear at the right of the formatted output. Two digits are displayed to the right of the decimal separator.

Scientific	Displays the number using scientific notation. One digit appears before the decimal separator and two after.
Yes/No	Displays No if the numeric expression is 0. Displays Yes for all other values.
True/False	Displays False if the numeric expression is 0. Displays True for all other values.
On/Off	Displays Off if the numeric expression is 0. Displays On for all other values.
Date/Time Formats	
Format	Description
General date	Displays the date and time. If there is no fractional part in the numeric expression, then only the date is displayed. If there is no integral part in the numeric expression, then only the time is displayed. Output is in the following form: 1/1/95 01:00:00 AM.
Long date	Displays a long date.
Medium date	Displays a medium date—prints out only the abbreviated name of the month.
Short date	Displays a short date.
Long time	Displays the long time. The default is: h:mm:ss.
Medium time	Displays the time using a 12-hour clock. Hours and minutes are displayed, and the AM/PM designator is at the end.
Short time	Displays the time using a 24-hour clock. Hours and minutes are displayed.
<p>User-Defined Formats In addition to the built-in formats, you can specify a user-defined format by using characters that have special meaning when used in a format expression. The following tables list the characters you can use for numeric, string, and date/time formats and explain their functions.</p>	
Numeric Formats	

Character	Meaning
Empty string	Displays the numeric expression as is, with no additional formatting.
0	This is a digit placeholder.
	Displays a number or a 0. If a number exists in the numeric expression in the position where the 0 appears, the number will be displayed. Otherwise, a 0 will be displayed. If there are more 0s in the format string than there are digits, the leading and trailing 0s are displayed without modification.
#	This is a digit placeholder.
	Displays a number or nothing. If a number exists in the numeric expression in the position where the number sign appears, the number will be displayed. Otherwise, nothing will be displayed. Leading and trailing 0s are not displayed.
.	This is the decimal placeholder.
	Designates the number of digits to the left of the decimal and the number of digits to the right. The character used in the formatted string depends on the decimal placeholder, as specified by your locale.
%	This is the percentage operator.
	The numeric expression is multiplied by 100, and the percent character is inserted in the same position as it appears in the user-defined format string.
,	This is the thousand separator.
	The common use for the thousands separator is to separate thousands from hundreds. To specify this use, the thousands separator must be surrounded by digit placeholders. Commas appearing before any digit placeholders are specified are just displayed. Adjacent commas with no digit placeholders specified between them and the decimal mean that the number should be divided by 1,000 for each adjacent comma in the format string. A comma immediately to the left of the decimal has the same function. The actual thousands separator character used depends on the character specified by your locale.
:E- E+ e- e+	These are the scientific notation operators, which display the number in scientific notation. At least one digit placeholder must exist to the left of E- , E+ , e- , or e+ . Any digit placeholders displayed to the left of E- , E+ , e- , or e+ determine the number

		of digits displayed in the exponent. Using E+ or e+ places a + in front of positive exponents and a – in front of negative exponents. Using E- or e- places a – in front of negative exponents and nothing in front of positive exponents.
	:	This is the time separator.
		Separates hours, minutes, and seconds when time values are being formatted. The actual character used depends on the character specified by your locale.
	/	This is the date separator.
		Separates months, days, and years when date values are being formatted. The actual character used depends on the character specified by your locale.
	:- + \$ () space	These are the literal characters you can display. To display any other character, you should precede it with a backslash or enclose it in quotes.
	\	This designates the next character as a displayed character.
		To display characters, precede them with a backslash. To display a backslash, use two backslashes. Double quotation marks can also be used to display characters. Numeric formatting characters, date/time formatting characters, and string formatting characters cannot be displayed without a preceding backslash.
	:"ABC"	Displays the text between the quotation marks, but not the quotation marks. To designate a double quotation mark within a format string, use two adjacent double quotation marks.
	*	This will display the next character as the fill character.
		Any empty space in a field will be filled with the specified fill character.
		Numeric formats can contain one to three parts. Each part is separated by a semicolon. If you specify one format, it applies to all values. If you specify two formats, the first applies to positive values and the second to negative values. If you specify three formats, the first applies to positive values, the second to negative values, and the third to 0s. If you include semicolons with no format between them, the format for positive values is used.
	String Formats	
	Char- acter	Meaning
	@	This is a character placeholder. Displays a character if one exists in the expression in the same position; otherwise, displays a space. Placeholders are filled from right to left unless the format string specifies left to right.

&	This is a character placeholder. Displays a character if one exists in the expression in the same position; otherwise, displays nothing. Placeholders are filled from right to left unless the format string specifies left to right.
<	This character forces lowercase. Displays all characters in the expression in lowercase.
>	This character forces uppercase. Displays all characters in the expression in uppercase.
!	This character forces placeholders to be filled from left to right. The default is right to left.
Date/Time Formats	
Char-acter	Meaning
c	Displays the date as dddd and the time as tttt . Only the date is displayed if no fractional part exists in the numeric expression. Only the time is displayed if no integral portion exists in the numeric expression.
d	Displays the day without a leading 0 (1–31).
dd	Displays the day with a leading 0 (01–31).
ddd	Displays the day of the week abbreviated (Sun–Sat).
dddd	Displays the day of the week (Sunday–Saturday).
ddddd	Displays the date as a short date.
dddddd	Displays the date as a long date.
w	Displays the number of the day of the week (1–7). Sunday is 1; Saturday is 7.
ww	Displays the week of the year (1–53).
m	Displays the month without a leading 0 (1–12). If m immediately follows h or hh, m is treated as minutes (0–59).
mm	Displays the month with a leading 0 (01–12). If mm immediately follows h or hh, mm is treated as minutes with a leading 0 (00–59).
mmm	Displays the month abbreviated (Jan–Dec).
mm-mm	Displays the month (January–December).

q	Displays the quarter of the year (1–4).
y	Displays the day of the year (1–366).
yy	Displays the year, not the century (00–99).
yyyy	Displays the year (1000–9999).
h	Displays the hour without a leading 0 (0–24).
hh	Displays the hour with a leading 0 (00–24).
n	Displays the minute without a leading 0 (0–59).
nn	Displays the minute with a leading 0 (00–59).
s	Displays the second without a leading 0 (0–59).
ss	Displays the second with a leading 0 (00–59).
tttt	Displays the time. A leading 0 is displayed if specified by your locale.
AM/ PM	Displays the time using a 12-hour clock. Displays an uppercase AM for time values before 12 noon. Displays an uppercase PM for time values after 12 noon and before 12 midnight.
am/pm	Displays the time using a 12-hour clock. Displays a lowercase am or pm at the end.
A/P	Displays the time using a 12-hour clock. Displays an uppercase A or P at the end.
a/p	Displays the time using a 12-hour clock. Displays a lowercase a or p at the end.
AMPM	Displays the time using a 12-hour clock. Displays the string s1159 for values before 12 noon and s2359 for values after 12 noon and before 12 midnight.
Example	<pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a# = 1199.234 msg1 = "Some general formats for '" & a# & "' are:" & crlf & crlf msg1 = msg1 & Format(a#, "General Number") & crlf msg1 = msg1 & Format(a#, "Currency") & crlf msg1 = msg1 & Format(a#, "Standard") & crlf msg1 = msg1 & Format(a#, "Fixed") & crlf msg1 = msg1 & Format(a#, "Percent") & crlf msg1 = msg1 & Format(a#, "Scientific") & crlf g1 = msg1 & Format(True, "Yes/No") & crlf msg1 = msg1 & Format(True, "True/False") & crlf </pre>

	<pre> msg1 = msg1 & Format(True,"On/Off") & crlf msg1 = msg1 & Format(a#,"0,0.00") & crlf msg1 = msg1 & Format(a#,"##,###,###.###") & crlf MsgBox msg1 da\$ = Date\$ msg1 = "Some date formats for '" & da\$ & "' are:" & crlf & crlf msg1 = msg1 & Format(da\$,"General Date") & crlf msg1 = msg1 & Format(da\$,"Long Date") & crlf msg1 = msg1 & Format(da\$,"Medium Date") & crlf msg1 = msg1 & Format(da\$,"Short Date") & crlf MsgBox msg1 ti\$ = Time\$ msg1 = "Some time formats for '" & ti\$ & "' are:" & crlf & crlf msg1 = msg1 & Format(ti\$,"Long Time") & crlf msg1 = msg1 & Format(ti\$,"Medium Time") & crlf msg1 = msg1 & Format(ti\$,"Short Time") & crlf MsgBox msg1 End Sub </pre>
<p>See Also</p>	<p>Str, Str\$ (on page 743) (functions); CStr (on page 386) (function).</p>
<p>Note</p>	<p>The default date/time formats are read from the [Intl] section of the win.ini file.</p>

FreeFile (function)

<p>Syntax</p>	<p>FreeFile()</p>
<p>Description</p>	<p>Returns an Integer containing the next available file number.</p>
<p>Comments</p>	<p>The number returned is suitable for use in the Open statement and will always be between 1 and 255 inclusive.</p>
<p>Example</p>	<p>This example assigns A to the next free file number and displays it in a dialog box.</p> <pre> Sub Main() a = FreeFile MsgBox "The next free file number is: " & a End Sub </pre>

See Also [FileAttr \(on page 512\)](#) (function); [Open \(on page 642\)](#) (statement).

Function...End Function (statement)

1. Must start with a letter.
2. May contain letters, digits, and the underscore character (`_`). Punctuation and type-declaration characters are not allowed.

The exclamation point (`!`) can appear within the name as long as it is not the last character, in which case it is interpreted as a type-declaration character.

3. Must not exceed 80 characters in length.
4. The call cannot end with a comma. For instance, using the above example, the following is not valid:

```
a = Test(1,,)
```

5. The call must contain the minimum number of parameters as required by the called function. For instance, using the above example, the following are invalid:

```
a = Test(,1) 'Only passes two out of three required parameters.
```

```
a = Test(1,2) 'Only passes two out of three required parameters.
```

Fv (function)

Syn-tax	Fv (Rate, Nper, Pmt,Pv,Due)	
De-scrip-tion	Calculates the future value of an annuity based on periodic fixed payments and a constant rate of interest.	
Com-ments	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans. The Fv function requires the following parameters:	
	Pa- ra- me- ter	Description
	Rate	Double representing the interest rate per period. Make sure that annual rates are normalized for monthly periods (divided by 12).

	NPer	Double representing the total number of payments (periods) in the annuity.
	Pmt	Double representing the amount of each payment per period. Payments are entered as negative values, whereas receipts are entered as positive values.
	Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, whereas in the case of a retirement annuity, the present value would be the amount of the fund.
	Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.
		Rate and NPer values must be expressed in the same units. If Rate is expressed as a percentage per month, then NPer must also be expressed in months. If Rate is an annual rate, then the NPer must also be given in years. Positive numbers represent cash received, whereas negative numbers represent cash paid out.
Example		<p>This example calculates the future value of 100 dollars paid periodically for a period of 10 years (120 months) at a rate of 10% per year (or .10/12 per month) with payments made on the first of the month. The value is displayed in a dialog box. Note that payments are negative values.</p> <pre> Sub Main() a# = Fv((.10/12),120,-100.00,0,1) MsgBox "Future value is: " & Format(a#,"Currency") End Sub </pre>
See Also		IRR (on page 568) (function); MIRR (on page 606) (function); Npv (on page 630) (function); Pv (on page 675) (function).

G

G

Get (statement)
GetAllSettings (function)
GetAttr (function)
GetObject (function)
GetSetting (function)

Global (statement)
GoSub (statement)
Goto (statement)
GroupBox (statement)

Get (statement)

Syn- tax	Get [#] filename, [recordnumber], variable	
De- scrip- tion	Retrieves data from a random or binary file and stores that data into the specified variable.	
Com- ments	The Get statement accepts the following parameters:	
	Para- meter	Description
	filenum- ber	Integer used by the Basic Control Engine to identify the file. This is the same number passed to the <code>Open</code> statement.
	record- num- ber	Long specifying which record is to be read from the file. For binary files, this number represents the first byte to be read starting with the beginning of the file (the first byte is 1). For random files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647. If the recordnumber parameter is omitted, the next record is read from the file (if no records have been read yet, then the first record in the file is read). When this parameter is omitted, the commas must still appear, as in the following example:
		<pre>Get #1,,recvar</pre>
		If recordnumber is specified, it overrides any previous change in file position specified with the <code>Seek</code> statement.
	vari- able	Variable into which data will be read. The type of the variable determines how the data is read from the file, as described below.
	With random files, a runtime error will occur if the length of the data being read exceeds the <code>reclen</code> parameter specified with the <code>Open</code> statement. If the length of the data being read is less	

	than the record length, the file pointer is advanced to the start of the next record. With binary files, the data elements being read are contiguous ^{3/4} the file pointer is never advanced.	
	Variable Types The type of the variable parameter determines how data will be read from the file. It can be any of the following types:	
	Variable Type	File Storage Description
	Integer	2 bytes are read from the file.
	Long	4 bytes are read from the file.
	String (variable-length)	In binary files, variable-length strings are read by first determining the specified string variable's length and then reading that many bytes from the file. For example, to read a string of eight characters:
		s\$ = String(8, " ") Get #1,,s\$
		In random files, variable-length strings are read by first reading a 2-byte length and then reading that many characters from the file.
	String (fixed-length)	Fixed-length strings are read by reading a fixed number of characters from the file equal to the string's declared length.
	Double	8 bytes are read from the file (IEEE format).
	Single	4 bytes are read from the file (IEEE format).
	Date	8 bytes are read from the file (IEEE double format).
	Boolean	2 bytes are read from the file. Nonzero values are True , and zero values are False .
	Variant	A 2-byte VarType is read from the file, which determines the format of the data that follows. Once the VarType is known, the data is read individually, as described above. With user-defined errors, after the 2-byte VarType , a 2-byte unsigned integer is read and assigned as the value of the user-defined error, followed by 2 additional bytes of information about the error. The exception is with strings, which are always preceded by a 2-byte string length.
	User-defined types	Each member of a user-defined data type is read individually In binary files, variable-length strings within user-defined types are read by first reading a 2-byte length followed by the string's content. This storage is different from variable-length strings outside of user-defined types. When reading user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.

	Arrays	Arrays cannot be read from a file using the Get statement.
	Objects	Object variables cannot be read from a file using the Get statement.
Example	<p>This example opens a file for random write, then writes ten records into the file with the values 10...50. Then the file is closed and reopened in random mode for read, and the records are read with the Get statement. The result is displayed in a message box.</p> <pre> Sub Main() Open "test.dat" For Random Access Write As #1 For x = 1 to 10 y = x * 10 Put #1,x,y Next x Close Open "test.dat" For Random Access Read As #1 msg1 = "" For y = 1 to 5 Get #1,y,x msg1 = msg1 & "Record " & y & ": " & x & Basic.Eoln\$ Next y Close MsgBox msg1 End Sub </pre>	
See Also	Open (on page 642) (statement); Put (on page 673) (statement); Input# (on page 559) (statement); Line Input# (on page 588) (statement); Input, Input\$ (on page 560) (functions)	

GetAllSettings (function)

Syntax	GetAllSettings(appname [,section])
Description	Returns all of the keys within the specified section, or all of the sections within the specified application from the system registry.
Comments	The <code>GetAllSettings</code> function takes the following named parameters:

	Parameter	Description
	appname	A String expression specifying the name of the application from which settings or keys will be returned.
	section	A String expression specifying the name of the section from which keys will be returned. If omitted, then all of the section names within appname will be returned.
The <code>GetAllSettings</code> function returns a Variant containing an array of strings.		
Example	<pre> Sub Main() Dim NewAppSettings() As Variant SaveSetting appname := "NewApp", section := "Startup", _ key := "Height", setting := 200 SaveSetting appname := "NewApp", section := "Startup _ ", key := "Width", setting := 320 GetAllSettings appname := "NewApp", _ section := "Startup ", resultarray := NewAppSettings For i = LBound(NewAppSettings) To UBound(NewAppSettings) NewAppSettings(i) = NewAppSettings(i) & "=" & _ GetSetting("NewApp", "Startup", NewAppSettings(i)) Next i r = SelectBox("Registry Settings","", NewAppSettings) End Sub </pre>	
See Also	GetSetting (on page 541) (function), DeleteSetting (on page 423) (statement), SaveSetting (on page 701) (statement)	
Notes	Under Win32, this statement operates on the system registry. All settings are read from the following entry in the system registry: <code>HKEY_CURRENT_USER\Software\BasicScript Program Settings\appname\section</code>	

GetAttr (function)

Syntax	GetAttr (filename\$)
--------	-----------------------------

De- scrip- tion	Returns an Integer containing the attributes of the specified file.		
Com- ments	The attribute value returned is the sum of the attributes set for the file. The value of each attribute is as follows:		
	Constant	Value	Includes
	EbNormal	0	Read-only files, archive files, subdirectories, and files with no attributes.
	EbReadOnly	1	Read-only files
	EbHidden	2	Hidden files
	EbSystem	4	System files
	EbVolume	8	Volume label
	EbDirectory	16	DOS subdirectories
	EbArchive	32	Files that have changed since the last backup
	EbNone	64	Files with no attributes
	<p>To determine whether a particular attribute is set, you can And the values shown above with the value returned by GetAttr . If the result is True , the attribute is set, as shown below:</p> <pre> Sub Main() Dim w As Integer w = GetAttr("sample.txt") If w And ebReadOnly Then MsgBox "This file is read-only." End Sub </pre>		
Exam- ple	<p>This example tests to see whether the file test.dat exists. If it does not, then it creates the file. The file attributes are then retrieved with the GetAttr function, and the result is displayed.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a() FileList a,"*.*)" Again: msg1 = "" r = SelectBox("Attribute Checker","Select File:",a) If r = -1 Then </pre>		

	<pre> End Else y% = GetAttr(a(x)) End If If y% = 0 Then msg1 = msg1 & "This file has no special attributes." & crlf If y% And ebReadOnly Then msg1 = msg1 & "The read-only bit is set." & crlf If y% And ebHidden Then msg1 = msg1 & "The hidden bit is set." & crlf If y% And ebSystem Then msg1 = msg1 & "The system bit is set." & crlf If y% And ebVolume Then msg1 = msg1 & "The volume bit is set." & crlf If y% And ebDirectory Then msg1 = msg1 & "The directory bit is set." & crlf If y% And ebArchive Then msg1 = msg1 & "The archive bit is set." MsgBox msg1 Goto Again End Sub </pre>
See Also	SetAttr (on page 715) (statement); FileAttr (on page 512) (function).

GetObject (function)

Syntax	GetObject (filename\$ [,class\$])
Description	Returns the object specified by filename\$ or returns a previously instantiated object of the given class\$.
Comments	This function is used to retrieve an existing OLE automation object, either one that comes from a file or one that has previously been instantiated.
	<p>The filename\$ argument specifies the full pathname of the file containing the object to be activated. The application associated with the file is determined by OLE at runtime. For example, suppose that a file called c:\docs\resume.doc was created by a word processor called wordproc.exe. The following statement would invoke wordproc.exe, load the file called c:\docs\resume.doc, and assign that object to a variable:</p> <pre> Dim doc As Object Set doc = GetObject("c:\docs\resume.doc") </pre>

	<p>To activate a part of an object, add an exclamation point to the filename followed by a string representing the part of the object that you want to activate. For example, to activate the first three pages of the document in the previous example:</p> <pre>Dim doc As Object Set doc = GetObject("c:\docs\resume.doc!P1-P3")</pre>		
	<p>The GetObject function behaves differently depending on whether the first parameter is omitted. The following table summarizes the different behaviors of GetObject :</p>		
	File-name\$\$	Class	GetObject Returns
	Omitted	Specified	Reference to an existing instance of the specified object. A runtime error results if the object is not already loaded.
	""	Specified	Reference to a new object (as specified by class\$). A runtime error occurs if an object of the specified class cannot be found. This is the same as CreateObject .
	Specified	Omitted	Default object from filename\$. The application to activate is determined by OLE based on the given filename.
	Specified	Specified	Object given by class\$ from the file given by filename\$. A runtime error occurs if an object of the given class cannot be found in the given file.
Example	<p>This first example instantiates the existing copy of Excel.</p> <pre>Sub Main() Dim Excel As Object Set Excel = GetObject(,"Excel.Application")</pre> <p>This second example loads the OLE server associated with a document.</p> <pre>Dim MyObject As Object Set MyObject = GetObject("c:\documents\resume.doc") End Sub</pre>		
See Also	<p>CreateObject (on page 368) (function); Object (on page 633) (data type).</p>		

GetSetting (function)

Syntax	<pre>GetSetting([appname], section, key[, default])</pre>
--------	---

De- scrip- tion	Retrieves an specific setting from the system registry.	
Com- ments	The <code>GetSetting</code> function has the following named parameters:	
	Para- me- ter	Description
	app- name	String expression specifying the name of the application from which the setting will be read.
	sec- tion	String expression specifying the name of the section within appname to be read.
	key	String expression specifying the name of the key within section to be read.
	de- fault	An optional String expression specifying the default value to be returned if the desired key does not exist in the system registry. If omitted, then an empty string is returned if the key doesn't exist.
Exam- ple	<pre> Sub Main() SaveSetting appname := "NewApp", section := "Startup", _ key := "Height", setting := 200 SaveSetting appname := "NewApp", section := "Startup", _ key := "Width", setting := 320 MsgBox GetSetting(appname := "NewApp", section := "Startup", _ key := "Height", default := "50") DeleteSetting "NewApp" ' Delete the NewApp key End Sub </pre>	
See Also	GetAllSettings (on page 537) (function), DeleteSetting (on page 423) (statement), SaveSetting (on page 701) (statement)	
Note	Under Win32, this statement operates on the system registry. All settings are read from the following entry in the system registry: <code>HKEY_CURRENT_USER\Software\BasicScript Program Settings\appname\section\key</code> On this platform, the appname parameter is not optional.	

Global (statement)

Description	See Public (on page 670) (statement).
-------------	---

GoSub (statement)

Syn- tax	GoSub label
De- scrip- tion	Causes execution to continue at the specified label.
Com- ments	Execution can later be returned to the statement following the GoSub by using the Return statement. The label parameter must be a label within the current function or subroutine. GoSub outside the context of the current function or subroutine is not allowed.
Exam- ple	<p>This example gets a name from the user and then branches to a subroutine to check the input. If the user clicks Cancel or enters a blank name, the program terminates; otherwise, the name is set to MICHAEL, and a message is displayed.</p> <pre> Sub Main() unname\$ = Ucase\$(InputBox\$("Enter your name:", "Enter Name")) GoSub CheckName MsgBox "I'm looking for MICHAEL, not " & unname\$ Exit Sub CheckName: If (unname\$ = "") Then GoSub BlankName ElseIf unname\$ = "MICHAEL" Then GoSub RightName Else GoSub OtherName End If Return BlankName: MsgBox "No name? Clicked Cancel? I'm shutting down." Exit Sub </pre>

	<pre> RightName: MsgBox "Hey, MIKE where have you been?" End OtherName: Return End Sub </pre>
See Also	Goto (on page 544) (statement); Return (on page 692) (statement).

Goto (statement)

The compiler will produce an error if label does not exist. The label must appear within the same subroutine or function as the **Goto**. Labels are identifiers that follow these rules:

1. Must begin with a letter.
2. May contain letters, digits, and the underscore character.
3. Must not exceed 80 characters in length.
4. Must be followed by a colon (:).

Labels are not case-sensitive.

GroupBox (statement)

Syn- tax	GroupBox X,Y,width,height,title\$ [,.Identifier]
De- scrip- tion	Defines a group box within a dialog box template.
Com- ments	This statement can only appear within a dialog box template (that is., between the Begin Dialog and End Dialog statements). The group box control is used for static display only; the user cannot interact with a group box control. Separator lines can be created using group box controls. This is accomplished by creating a group box that is wider than the width of the dialog box and extends below the bottom of the dialog box; three sides of the group box are not visible.
	If title\$ is a zero-length string, then the group box is drawn as a solid rectangle with no title. The <code>GroupBox</code> statement requires the following parameters:

	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	title\$	String containing the label of the group box. If title\$ is a zero-length string, then no title will appear.
	.Identifier	Optional parameter that specifies the name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words of title\$ are used.
Example	<p>This example shows the <code>GroupBox</code> statement being used both for grouping and as a separator line.</p> <pre> Sub Main() Begin Dialog OptionsTemplate 16,32,128,84,"Options" GroupBox 4,4,116,40,"Window Options" CheckBox 12,16,60,8,"Show &Toolbar",.ShowToolbar CheckBox 12,28,68,8,"Show &Status Bar",.ShowStatusBar GroupBox -12,52,152,48," ",.SeparatorLine OKButton 16,64,40,14,.OK CancelButton 68,64,40,14,.Cancel End Dialog Dim OptionsDialog As OptionsTemplate Dialog OptionsDialog End Sub </pre>	
See Also	<p>CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin (on page 348) Dialog (on page 348) (statement), PictureButton (on page 659) (statement).</p>	

H

H

HelpButton (statement)
Hex, Hex\$ (function)
HLine (statement)
Hour (function)
HPage (statement)
HScroll (statement)
HWND (object)
HWND.Value (property)

HelpButton (statement)

Syntax	<code>HelpButton x,y,width,height,HelpFileName\$,HelpContext, [,Identifier]</code>	
Description	Defines a help button within a dialog template.	
Comments	This statement can only appear within a dialog box template (i.e., between the <code>Begin Dialog</code> and <code>End Dialog</code> statements). The <code>HelpButton</code> statement takes the following parameters:	
	Parameter	Description
	x,y	Integer position of the control (in dialog units) static to the upper left corner of the dialog box.
	width,height	Integer dimensions of the control in dialog units.
	HelpFileName\$	String expression specifying the name of the help file to be invoked when the button is selected.
	HelpContext	Long expression specifying the ID of the topic within <code>HelpFileName\$</code> containing context-sensitive help.

	.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).
	When the user selects a help button, the associated help file is located at the indicated topic. Selecting a help button does not remove the dialog. Similarly, no actions are sent to the dialog procedure when a help button is selected. When a help button is present within a dialog, it can be automatically selected by pressing the help key (F1 on most platforms).	
Example	<pre> Sub Main() Begin Dialog HelpDialogTemplate ,,180,96,"Untitled" OKButton 132,8,40,14 CancelButton 132,28,40,14 HelpButton 132,48,40,14,"", 10 Text 16,12,88,12,"Please click ""Help"".",>.Text1 End Dialog Dim HelpDialog As HelpDialogTemplate Dialog HelpDialog End Sub </pre>	
See Also	CancelButton (on page 365) (statement), CheckBox (on page 360) (statement), ComboBox (on page 373) (statement), Dialog (on page 424) (function), Dialog (on page 426) (statement), DropListBox (on page 458) (statement), GroupBox (on page 544) (statement), ListBox (on page 591) (statement), OKButton (on page 638) (statement), OptionButton (on page 652) (statement), OptionGroup (on page 653) (statement), Picture (on page 657) (statement), PushButton (on page 671) (statement), Text (on page 752) (statement), Begin Dialog (on page 348) (statement), PictureButton (on page 659) (statement)	

Hex, Hex\$ (functions)

Syntax	Hex[\$] (number)
Description	Returns a String containing the hexadecimal equivalent of number.

Com- ments	Hex\$ returns a String , whereas Hex returns a String variant. The returned string contains only the number of hexadecimal digits necessary to represent the number, up to a maximum of eight.
	The number parameter can be any type but is rounded to the nearest whole number before converting to hex. If the passed number is an integer, then a maximum of four digits are returned; otherwise, up to eight digits can be returned. The number parameter can be any expression convertible to a number. If number is Null , then Null is returned. Empty is treated as 0.
Exam- ple	<p>This example accepts a number and displays the decimal and hexadecimal equivalent until the input number is 0 or invalid.</p> <pre> Sub Main() Do xs\$ = InputBox("Enter a number to convert:", "Hex Convert") x = Val(xs\$) If x <> 0 Then MsgBox "Decimal: " & x & " Hex: " & Hex(x) Else MsgBox "Goodbye." End If Loop While x <> 0 End Sub </pre>
See Also	Oct , Oct\$ (on page 637) (functions).

HLine (statement)

Syn- tax	HLine [lines]
De- scrip- tion	Scrolls the window with the focus left or right by the specified number of lines.
Com- ments	The lines parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled right by one line.
Exam- ple	This example scrolls the Notepad window to the left by three "amounts." Each "amount" is equivalent to clicking the right arrow of the horizontal scroll bar once.

	<pre>Sub Main() AppActivate "Notepad" HLine 3 'Move 3 lines in. End Sub</pre>
See Also	HPage (on page 549) (statement); HScroll (on page 550) (statement).

Hour (function)

Syntax	Hour (time)
Description	Returns the hour of the day encoded in the specified time parameter.
Comments	The value returned is as an Integer between 0 and 23 inclusive. The time parameter is any expression that converts to a Date .
Example	<p>This example takes the current time; extracts the hour, minute, and second; and displays them as the current time.</p> <pre>Sub Main() MsgBox "It is now hour " & Hour(Time) & " of today." End Sub</pre>
See Also	Day (on page 401) (function); Minute (on page 606) (function); Second (on page 705) (function); Month (on page 610) (function); Year (on page 797) (function); Weekday (on page 779) (function); DatePart (on page 398) (function).

HPage (statement)

Syntax	HPage [pages]
Description	Scrolls the window with the focus left or right by the specified number of pages.
Comments	The pages parameter is an Integer specifying the number of pages to scroll. If this parameter is omitted, then the window is scrolled right by one page.

Exam- ple	<p>This example scrolls the Notepad window to the left by three "amounts." Each "amount" is equivalent to clicking within the horizontal scroll bar on the right side of the thumb mark.</p> <pre> Sub Main() AppActivate "Notepad" HPage 3 'Move 3 pages down. End Sub </pre>
See Also	<p>HLine (on page 548) (statement); HScroll (on page 550) (statement).</p>

HScroll (statement)

Syn- tax	<p>HScroll percentage</p>
De- scrip- tion	<p>Sets the thumb mark on the horizontal scroll bar attached to the current window.</p>
Com- ments	<p>The position is given as a percentage of the total range associated with that scroll bar. For example, if the percentage parameter is 50, then the thumb mark is positioned in the middle of the scroll bar.</p>
Exam- ple	<p>This example centers the thumb mark on the horizontal scroll bar of the Notepad window.</p> <pre> Sub Main() AppActivate "Notepad" HScroll 50 'Jump to the middle of the document. End Sub </pre>
See Also	<p>HLine (on page 548) (statement); HPage (on page 549) (statement).</p>

HWND (object)

Syntax	<p>Dim name As HWND</p>
De- scrip- tion	<p>A data type used to hold window objects.</p>

Com- ments	<p>This data type is used to hold references to physical windows in the operating environment. The following commands operate on HWND objects:</p>			
	WinActivate	WinClose	WinFind	WinList
	WinMaximize	WinMinimize	WinMove	WinRestore
	WinSize			
	<p>The above language elements support both string and HWND window specifications.</p>			
Exam- ple	<p>This example activates the "Main" MDI window within Program Manager.</p> <pre> Sub Main() Dim ProgramManager As HWND Dim ProgramManagerMain As HWND Set ProgramManager = WinFind("Program Manager") If ProgramManager Is Not Nothing Then WinActivate ProgramManager WinMaximize ProgramManager Set ProgramManagerMain = WinFind("Program Manager Main") If ProgramManagerMain Is Not Nothing Then WinActivate ProgramManagerMain WinRestore ProgramManagerMain Else MsgBox "Your Program Manager doesn't have a Main group." End If Else MsgBox "Program Manager is not running." End If End Sub </pre>			
See Al- so	<p>HWND.Value (on page 551) (property); WinFind (on page 785) (function); WinActivate (on page 782) (statement).</p>			

HWND.Value (property)

Syn- tax	window .Value
-------------	----------------------

De- scrip- tion	The default property of an HWND object that returns a VARIANT containing a HANDLE to the physical window of an HWND object variable.
Com- ments	The .Value property is used to retrieve the operating environment-specific value of a given HWND object. The size of this value depends on the operating environment in which the script is executing and thus should always be placed into a VARIANT variable. This property is read-only.
Exam- ple	<p>This example displays a dialog box containing the class name of Program Manager's Main window. It does so using the .Value property, passing it directly to a Windows external routine.</p> <pre style="background-color: #f0f0f0; padding: 10px;"> Declare Sub GetClassName Lib "user" (ByVal Win%,ByVal ClsName\$, ByVal ClsNameLen%) Sub Main() Dim ProgramManager As HWND Set ProgramManager = WinFind("Program Manager") ClassName\$ = Space(40) GetClassName ProgramManager.Value,ClassName\$,Len(ClassName\$) MsgBox "The program classname is: " & ClassName\$ End Sub </pre>
See Also	HWND (on page 550) (object).
Notes	Under Windows, this value is an Integer .

|

|

If...Then...Else (statement)
IIf (function)
IMEStatus (function)
Imp (operator)
Input# (statement)
Input, Input\$, InputB, InputB\$ (functions)

InputBox, InputBox\$ (functions)
InStr, InStrB (functions)
Int (function)
Integer (data type)
IPmt (function)
IRR (function)
Is (operator)
IsDate (function)
IsEmpty (function)
IsError (function)
IsMissing (function)
IsNull (function)
IsNumeric (function)
IsObject (function)
IsWebSpaceSession (function)
Item\$ (function)
ItemCount (function)

If...Then...Else (statement)

Syn- tax 1	<code>If condition Then statements [Else else_statements]</code>
Syn- tax 2	<code>If condition Then [statements] [ElseIf else_condition Then [elseif_statements]] [Else [else_statements]] End If</code>
De- scrip- tion	Conditionally executes a statement or group of statements.
Com- ments	The single-line conditional statement (syntax 1) has the following parameters:

	Parameter	Description
	condition	Any expression evaluating to a Boolean value.
	statements	One or more statements separated with colons. This group of statements is executed when condition is TRUE.
	else_state- ments	One or more statements separated with colons. This group of statements is executed when condition is FALSE.
The multiline conditional statement (syntax 2) has the following parameters:		
	Parameter	Description
	condition	Any expression evaluating to a Boolean value.
	statements	One or more statements to be executed when condition is True .
	else_condi- tion	Any expression evaluating to a Boolean value. The else_condition is evaluated if condition is False .
	elseif_state- ments	One or more statements to be executed when condition is False and else_condi- tion is True .
	else_state- ments	One or more statements to be executed when both condition and else_condition are False .
There can be as many Elseif conditions as required.		
Exam- ple	<p>This example inputs a name from the user and checks to see whether it is MICHAEL or MIKE using three forms of the If...Then...Else statement. It then branches to a statement that displays a welcome message depending on the user's name.</p> <pre> Sub Main() unname\$ = UCase(InputBox("Enter your name:","Enter Name")) If unname\$ = "MICHAEL" Then GoSub MikeName If unname\$ = "MIKE" Then GoSub MikeName Exit Sub End If If unname\$ = "" Then MsgBox "Since you don't have a name, I'll call you MIKE!" unname\$ = "MIKE" GoSub MikeName ElseIf unname\$ = "MICHAEL" Then </pre>	

	<pre> GoSub MikeName Else GoSub OtherName End If Exit Sub MikeName: MsgBox "Hello, MICHAEL!" Return OtherName: MsgBox "Hello, " & uname\$ & "!" Return End Sub </pre>
See Also	Choose (on page 362) (function); Switch (on page 744) (function); IIf (on page 555) (function); Select...Case (on page 708) (statement).

IIf (function)

Syntax	IIf (condition,TrueExpression,FalseExpression)
Description	Returns TrueExpression if condition is True ; otherwise, returns FalseExpression.
Comments	Both expressions are calculated before IIf returns. The IIf function is shorthand for the following construct: <pre> If condition Then variable = TrueExpression Else variable = FalseExpression End If </pre>
Example	<pre> Sub Main() s\$ = "Car" MsgBox "You have a " & IIf(s\$ = "Car","nice car. ","nice non-car.") End Sub </pre>
See Also	Choose (on page 362) (function); Switch (on page 744) (function); If...Then...Else (on page 553) (statement); Select...Case (on page 708) (statement).

IMEStatus (function)

Syntax	<code>IMEStatus[()]</code>		
Description	Returns the current status of the input method editor.		
Comments	The <code>IMEStatus</code> function returns one of the following constants for Japanese locales:		
	Constant	Value	Description
	<code>ebIMENoOp</code>	0	IME not installed.
	<code>ebIMEOn</code>	1	IME on.
	<code>ebIMEOff</code>	2	IME off.
	<code>ebIMEDisabled</code>	3	disabled
	<code>ebIMEHiragana</code>	4	Hiragana double-byte character.
	<code>ebIMEKatakanaDbl</code>	5	Katakana double-byte characters.
	<code>ebIMEKatakanaSng</code>	6	Katakana single-byte characters.
	<code>ebIMEAlphaDbl</code>	7	Alphanumeric double-byte characters.
	<code>ebIMEAlphaSng</code>	8	Alphanumeric single-byte characters.
	For Chinese locales, one of the following constants are returned:		
	Constant	Value	Description
	<code>ebIMENoOp</code>	0	IME not installed.
	<code>ebIMEOn</code>	1	IME on.
	<code>ebIMEOff</code>	2	IME off.
	For Korean locales, this function returns a value with the first 5 bits having the following meaning:		
	Bit	If not set (or 0)	If set (or 1)
	Bit 0	IME not installed	IME installed
	Bit 1	IME disabled	IME enabled

	Bit 2	English mode	Hangeul mode
	Bit 3	Banja mode (single-byte)	Junja mode (double-byte)
	Bit 4	Normal mode	Hanja conversion mode
	Note: You can test for the different bits using the And operator as follows:		
	a = IMEStatus() If a And 1 Then ... 'Test for bit 0 If a And 2 Then ... 'Test for bit 1 If a And 4 Then ... 'Test for bit 2 If a And 8 Then ... 'Test for bit 3 If a And 16 Then ... 'Test for bit 4		
	This function always returns 0 if no input method editor is installed.		
Example	<pre> This example retrieves the IMEStatus and displays the results. Sub Main() a = IMEStatus() Select case a Case 0 MsgBox "IME not installed." Case 1 MsgBox "IME on." Case 2 MsgBox "IME off." End Select End Sub </pre>		
See Also	Constants (on page 380) (topic)		

Imp (operator)

Syntax	expression1 <code>Imp</code> expression2
Description	Performs a logical or binary implication on two expressions.

Com-ments	If both expressions are either Boolean , Boolean variants, or Null variants, then a logical implication is performed as follows:					
	If the first expres- sion is	and the second ex- pression is	then the result is			
	TRUE	TRUE	TRUE			
	TRUE	FALSE	FALSE			
	TRUE	NULL	NULL			
	FALSE	TRUE	TRUE			
	FALSE	FALSE	TRUE			
	FALSE	NULL	TRUE			
	NULL	TRUE	TRUE			
	NULL	FALSE	NULL			
	NULL	NULL	NULL			
	Binary Implication If the two expressions are Integer , then a binary implication is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long and a binary implication is then performed, returning a Long result. Binary implication forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions, according to the following table:					
	1	Imp	1	=	1	Example
	0	Imp	1	=	1	5 01101001
	1	Imp	0	=	0	6 10101010
	0	Imp	0	=	1	Imp 10111110
Exam-ple	<p>This example compares the result of two expressions to determine whether one implies the other.</p> <pre> Sub Main() a = 10 : b = 20 : c = 30 : d = 40 If (a < b) Imp (c < d) Then MsgBox "a is less than b implies that c is less than d." Else MsgBox "a is less than b does not imply that c is less than d." End If End Sub </pre>					

	<pre> End If If (a < b) Imp (c > d) Then MsgBox "a is less than b implies that c is greater than d." Else MsgBox "a is less than b does not imply that c is greater than d." End If End Sub </pre>
See	Operator Precedence (on page 648) (topic); Or (on page 654) (operator); Xor (on page 795)
Also	(operator) ; Eqv (on page 489) (operator); And (operator) . (on page 309)

Input# (statement)

Each variable must be type-matched to the data in the file. For example, a **String** variable must be matched to a string in the file. The following parsing rules are observed while reading each variable in the variable list:

1. Leading white space is ignored (spaces and tabs).
2. When reading **String** variables, if the first character on the line is a quotation mark, then characters are read up to the next quotation mark or the end of the line, whichever comes first. Blank lines are read as empty strings. If the first character read is not a quotation mark, then characters are read up to the first comma or the end of the line, whichever comes first. String delimiters (quotes, comma, end-of-line) are not included in the returned string.
3. When reading numeric variables, scanning of the number stops when the first nonnumber character (such as a comma, a letter, or any other unexpected character) is encountered. Numeric errors are ignored while reading numbers from a file. The resultant number is automatically converted to the same type as the variable into which the value will be placed. If there is an error in conversion, then 0 is stored into the variable.

`octaldigits [!|#|%|&|@]` After reading the number, input is skipped up to the next delimiter—a comma, an end-of-line, or an end-of-file. Numbers must adhere to any of the following syntaxes: `[-+]
+]digits[digits][E[-|+]digits][!|#|%|&|@] &Hhexdigits[!|#|%|&] &[O]`

4. When reading `Boolean` variables, the first character must be #; otherwise, a runtime error occurs. If the first character is #, then input is scanned up to the next delimiter (a comma, an end-of-line, or an end-of-file). If the input matches #FALSE#, then FALSE is stored in the **Boolean** ; otherwise TRUE is stored.
5. When reading `Date` variables, the first character must be #; otherwise, a runtime error occurs. If the first character is #, then the input is scanned up to the next delimiter (a comma, an end-of-line, or

an end-of-file). If the input ends in a # and the text between the #'s can be correctly interpreted as a date, then the date is stored; otherwise, December 31, 1899, is stored.

Normally, dates that follow the universal date format are input from sequential files. These dates use this syntax: #YYYY-MM-DD HH:MM:SS# where YYYY is a year between 100 and 9999, MM is a month between 1 and 12, DD is a day between 1 and 31, HH is an hour between 0 and 23, MM is a minute between 0 and 59, and SS is a second between 0 and 59.

6. When reading `variant` variables, if the data begins with a quotation mark, then a string is read consisting of the characters between the opening quotation mark and the closing quotation mark, end-of-line, or end-of-file.

If the input does not begin with a quotation mark, then input is scanned up to the next comma, end-of-line, or end-of-file and a determination is made as to what data is being represented. If the data cannot be represented as a number, **Date**, **Error**, **Boolean**, or **Null**, then it is read as a string. The following table describes how special data is interpreted as variants:

7. End-of-line is interpreted as either a single line feed, a single carriage return, or a carriage-return/line-feed pair. Thus, text files from any platform can be interpreted using this command.

The `filenumber` parameter is a number that is used by The Basic Control Engine to refer to the open file, the number passed to the **Open** statement. The `filenumber` must reference a file opened in **Input** mode. It is good practice to use the **Write** statement to write date elements to files read with the **Input** statement to ensure that the variable list is consistent between the input and output routines.

Input, Input\$, InputB, InputB\$ (functions)

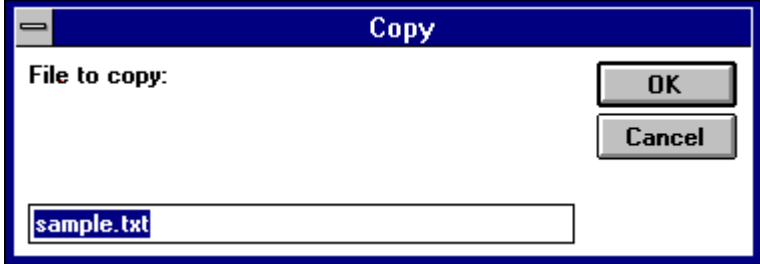
Syntax	<code>Input[\$](numchars,[#]filenumber) InputB[\$](numbytes,[#]filenumber)</code>	
Description	Returns a specified number of characters or bytes read from a given sequential file.	
Comments	The functions return the following.	
	Functions	Return
	Input\$ and InputB\$	String
	Input and InputB	String variant.
	The following parameters are required:	

	Parameter	Description
	numchars	Integer containing the number of characters to be read from the file.
	numbytes	Integer containing the number of bytes to be read from the file.
	filenumber	Integer referencing a file opened in either Input or Binary mode. This is the same number passed to the <code>Open</code> statement.
Functions are used to read the following.		
	Functions	Used to read:
	InputB and InputB\$	Byte data from a file.
	Input and Input\$	All characters, including spaces and end-of-lines. Null characters are ignored.
Example	<pre> This example opens the autoexec.bat file and displays it in a 'dialog box. Const crlf = Chr\$(13) & Chr\$(10) Sub Main() x& = FileLen("c:\autoexec.bat") If x& > 0 Then Open "c:\autoexec.bat" For Input As #1 Else MsgBox "File not found or empty." Exit Sub End If If x& > 80 Then ins = Input(80,#1) Else ins = Input(x,#1) End If Close MsgBox "File length: " & x& & crlf & ins End Sub </pre>	

See Also	Open (on page 642) (statement); Get (on page 535) (statement); Input# (on page 559) (statement); Line Input# (on page 588) (statement).
----------	---

InputDialog, InputBox\$ (functions)

Syntax	<code>InputDialog\$(prompt [, [title] [, [default] [, [xpos],[ypos] [,helpfile,context]]])</code>	
Description	Displays a dialog box with a text box into which the user can type.	
Comments	The content of the text box is returned as a String (in the case of InputBox\$) or as a String variant (in the case of InputBox). A zero-length string is returned if the user selects Cancel. The InputBox/InputBox\$ functions take the following named parameters:	
	Parameter	Description
	prompt	Text to be displayed above the text box. The prompt parameter can contain multiple lines, each separated with an end-of-line (a carriage return, line feed, or carriage-return/line-feed pair). A runtime error is generated if prompt is Null.
	title	Caption of the dialog box. If this parameter is omitted, then no title appears as the dialog box's caption. A runtime error is generated if title is Null.
	default	Default response. This string is initially displayed in the text box. A runtime error is generated if default is Null.
	xpos, ypos	Integer coordinates, given in twips (twentieths of a point), specifying the upper left corner of the dialog box static to the upper left corner of the screen. If the position is omitted, then the dialog box is positioned on or near the application executing the script.

	helpfile	Name of the file containing context-sensitive help for this dialog. If this parameter is specified, then context must also be specified.
	context	Number specifying the ID of the topic within helpfile for this dialog's help. If this parameter is specified, then helpfile must also be specified.
	<p>You can type a maximum of 255 characters into <code>InputBox</code>. If both the helpfile and context parameters are specified, then a Help button is added in addition to the OK and Cancel buttons. Context-sensitive help can be invoked by selecting this button or using the help key (F1 on most platforms). Invoking help does not remove the dialog. When Cancel is selected, an empty string is returned. An empty string is also returned when the user selects the OK button with no text in the input box. Thus, it is not possible to determine the difference between these two situations. If you need to determine the difference, you should create a user-defined dialog or use the <code>AskBox</code> function.</p>	
Example	<pre> Sub Main() s\$ = InputBox\$("File to copy:", "Copy", "sample.txt") End Sub </pre> 	
See Also	<p>MsgBox (on page 617) (statement), AskBox, AskBox\$ (on page 333) (functions), AskPassword, AskPassword\$ (on page 335) (functions), OpenFileName\$ (on page 646) (function), SaveFileName\$ (on page 699) (function), SelectBox (on page 709) (function), AnswerBox (on page 310) (function)</p>	

InStr, InStrB (functions)

Syntax	<code>InStr([start,] search, find [,compare])</code> <code>InStrB([start,] search, find [,compare])</code>
--------	--

De- scrip- tion	Returns the first character position of string find within string search.	
Com- ments	The <code>InStr</code> function takes the following parameters:	
	Para- meter	Description
	start	Integer specifying the character position where searching begins. The start parameter must be between 1 and 32767. If this parameter is omitted, then the search starts at the beginning (start = 1).
	search	Text to search. This can be any expression convertible to a String .
	find	Text for which to search. This can be any expression convertible to a String .
	com- pare	Integer controlling how string comparisons are performed:
		0
		String comparisons are case-sensitive.
		1
		String comparisons are case-insensitive.
		Any other value
		A runtime error is produced.
		If this parameter is omitted, then string comparisons use the current Option Compare setting. If no Option Compare statement has been encountered, then Binary is used (i.e., string comparisons are case-sensitive).
	If the string is found, then its character position within search is returned, with 1 being the character position of the first character.	
	<p>The <code>InStr</code> and <code>InStrB</code> functions observe the following additional rules:</p> <ul style="list-style-type: none"> • If either search or find is NULL, then NULL is returned. • If the compare parameter is specified, then start must also be specified. In other words, if there are three parameters, then it is assumed that these parameters correspond to start, search, and find. • A runtime error is generated if start is NULL. • A runtime error is generated if compare is not 0 or 1. • If search is Empty, then 0 is returned. 	

	<ul style="list-style-type: none"> • If find is Empty, then start is returned. If start is greater than the length of search, then 0 is returned. • A runtime error is generated if start is less than or equal to 0.
	<p>The <code>InStr</code> and <code>InStrB</code> functions operate on character and byte data respectively. The <code>InStr</code> function interprets the start parameter as a character, performs a textual comparisons, and returns a character position. The <code>InStrB</code> function, on the other hand, interprets the start parameter as a byte position, performs binary comparisons, and returns a byte position. On SBCS platforms, the <code>InStr</code> and <code>InStrB</code> functions are identical.</p>
Exam- ple	<p>This example checks to see whether one string is in another and, if it is, then it copies the string to a variable and displays the result.</p>
	<pre> Sub Main() a\$ = "This string contains the name Stuart and other characters." x% = InStr(1, a\$, "Stuart", 1) If x% <> 0 Then b\$ = Mid(a\$, x%, 6) MsgBox b\$ & " was found." Exit Sub Else MsgBox "Stuart not found." End If End Sub </pre>
See Also	<p>Mid, Mid\$ (on page 603) (functions); Option Compare (on page 649) (statement); Item\$ (on page 576) (function); Word\$ (on page 792) (function); Line\$ (on page 589) (function).</p>

Int (function)

Syn- tax	Int (number)
De- scrip- tion	Returns the integer part of number.
Com- ments	This function returns the integer part of a given value by returning the first integer less than the number. The sign is preserved. The Int function returns the same type as number, with the following exceptions:

	<ul style="list-style-type: none"> • If number is Empty , then an Integer variant of value 0 is returned. • f number is a String , then a Double variant is returned. • If number is Null , then a Null variant is returned.
Example	<p>This example extracts the integer part of a number.</p> <pre>Sub Main() a# = -1234.5224 b% = Int(a#) MsgBox "The integer part of -1234.5224 is: " & b% End Sub</pre>
See Also	Fix (on page 521) (function); Clnt (on page 364) (function).

Integer (data type)

Syntax	Integer
Description	A data type used to declare whole numbers with up to four digits of precision.
Comments	<p>Integer variables are used to hold numbers within the following range:</p> <pre>-32768 <= integer <= 32767</pre> <p>Internally, integers are 2-byte short values. Thus, when appearing within a structure, integers require 2 bytes of storage. When used with binary or random files, 2 bytes of storage are required. When passed to external routines, Integer values are sign-extended to the size of an integer on that platform (either 16 or 32 bits) before pushing onto the stack. The type-declaration character for Integer is % .</p>
See Also	Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Long (on page 598) (data type), Object (on page 633) (data type), Single (on page 718) (data type), String (on page 742) (data type), Variant (on page 771) (data type), Boolean (on page 351) (data type), DefType (on page 421) (statement), Clnt (on page 364) (function).

IPmt (function)

Syn-tax	IPmt (Rate, Per, Nper, Pv, Fv, Due)	
De-scrip-tion	Returns the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.	
Com-ments	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages, monthly savings plans, and retirement plans. The following table describes the different parameters:	
	Pa- ra- me- ter	Description
	Rate	Double representing the interest rate per period. If the payment periods are monthly, be sure to divide the annual interest rate by 12 to get the monthly rate.
	Per	Double representing the payment period for which you are calculating the interest payment. If you want to know the interest paid or received during period 20 of an annuity, this value would be 20.
	Nper	Double representing the total number of payments in the annuity. This is usually expressed in months, and you should be sure that the interest rate given above is for the same period that you enter here.
	Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan because that is the amount of cash you have in the present. In the case of a retirement plan, this value would be the current value of the fund because you have a set amount of principal in the plan.
	Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be zero because you will have paid it off. In the case of a savings plan, the future value would be the balance of the account after all payments are made.
	Due	Integer indicating when payments are due. If this parameter is 0, then payments are due at the end of each period (usually, the end of the month). If this value is 1, then payments are due at the start of each period (the beginning of the month).
	Rate and Nper must be in expressed in the same units. If Rate is expressed in percentage paid per month, then Nper must also be expressed in months. If Rate is an annual rate, then the period given in Nper should also be in years or the annual Rate should be divided by 12 to obtain a	

	monthly rate. If the function returns a negative value, it represents interest you are paying out, whereas a positive value represents interest paid to you.
Example	<p>This example calculates the amount of interest paid on a \$1,000.00 loan financed over 36 months with an annual interest rate of 10%. Payments are due at the beginning of the month. The interest paid during the first 10 months is displayed in a table.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() msg1 = "" For x = 1 to 10 ipm# = IPmt((.10/12),x,36,1000,0,1) msg1 = msg1 & Format(x,"00") & " : " & Format(ipm#," 0,0.00") & crlf Next x MsgBox msg1 End Sub </pre>
See Also	NPer (on page 629) (function); Pmt (on page 661) (function); PPmt (on page 663) (function); Rate (on page 680) (function).

IRR (function)

Syntax	IRR (ValueArray(),Guess)				
Description	Returns the internal rate of return for a series of periodic payments and receipts.				
Comments	The internal rate of return is the equivalent rate of interest for an investment consisting of a series of positive and/or negative cash flows over a period of regular intervals. It is usually used to project the rate of return on a business investment that requires a capital investment up front and a series of investments and returns on investment over time. The IRR function requires the following parameters:				
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ValueArray()</td> <td>Array of Double numbers that represent payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one</td> </tr> </tbody> </table>	Parameter	Description	ValueArray()	Array of Double numbers that represent payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one
Parameter	Description				
ValueArray()	Array of Double numbers that represent payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one				

	negative value to indicate the initial investment (negative value) and the amount earned by the investment (positive value).
Guess	Double containing your guess as to the value that the IRR function will return. The most common guess is .1 (10 percent).
	The value of IRR is found by iteration. It starts with the value of Guess and cycles through the calculation adjusting Guess until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, IRR fails, and the user must pick a better guess.
Example	<p>This example illustrates the purchase of a lemonade stand for \$800 and a series of incomes from the sale of lemonade over 12 months. The projected incomes for this example are generated in two For...Next Loops, and then the internal rate of return is calculated and displayed. (Not a bad investment!)</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim valu#(12) valu(1) = -800 'Initial investment msg1 = valu#(1) & ", " 'Calculate the second through fifth months' sales. For x = 2 To 5 valu(x) = 100 + (x * 2) msg1 = msg1 & valu(x) & ", " Next x 'Calculate the sixth through twelfth months' sales. For x = 6 To 12 valu(x) = 100 + (x * 10) msg1 = msg1 & valu(x) & ", " Next x 'Calculate the equivalent investment return rate. retrn# = IRR(valu,.1) msg1 = "The values: " & crlf & msg1 & crlf & crlf MsgBox msg1 & "Return rate: " & Format(retrn#,"Percent") End Sub </pre>
See Also	Fv (on page 533) (function); MIRR (on page 606) (function); Npv (on page 630) (function); Pv (on page 675) (function).

Is (operator)

Syntax	object Is [object Nothing]
Description	Returns True if the two operands refer to the same object; returns False otherwise.
Comments	<p>This operator is used to determine whether two object variables refer to the same object. Both operands must be object variables of the same type (i.e., the same data object type or both of type Object). The Nothing constant can be used to determine whether an object variable is uninitialized:</p> <pre data-bbox="305 747 1419 800">If MyObject Is Nothing Then MsgBox "MyObject is uninitialized."</pre> <p>Uninitialized object variables reference no object.</p>
Example	<p>This function inserts the date into a Microsoft Word document.</p> <pre data-bbox="305 936 1419 1520">Sub InsertDate(ByVal WinWord As Object) If WinWord Is Nothing Then MsgBox "Object variant is not set." Else WinWord.Insert Date\$ End If End Sub Sub Main() Dim WinWord As Object On Error Resume Next WinWord = CreateObject("word.basic") InsertDate WinWord End Sub</pre>
See Also	Operator Precedence (on page 648) (topic); Like (on page 587) (operator).
Platform(s)	All.

Notes	<p>When comparing OLE automation objects, the Is operator will only return True if the operands reference the same OLE automation object. This is different from data objects. For example, the following use of Is (using the object class called excel.application) returns True :</p> <pre data-bbox="305 352 1421 583"> Dim a As Object Dim b As Object a = CreateObject("excel.application") b = a If a Is b Then Beep </pre>
	<p>The following use of Is will return False , even though the actual objects may be the same:</p> <pre data-bbox="305 657 1421 888"> Dim a As Object Dim b As Object a = CreateObject("excel.application") b = GetObject(,"excel.application") If a Is b Then Beep </pre> <p>The Is operator may return False in the above case because, even though a and b reference the same object, they may be treated as different objects by OLE 2.0 (this is dependent on the OLE 2.0 server application).</p>

IsDate (function)

Syntax	IsDate (expression)
De- scrip- tion	Returns True if expression can be legally converted to a date; returns False otherwise.
Exam- ple	<pre data-bbox="326 1413 1421 1791"> Sub Main() Dim a As Variant Retry: a = InputBox("Enter a date.", "Enter Date") If IsDate(a) Then MsgBox Format(a, "long date") Else MsgBox "Not quite, please try again!" Goto Retry </pre>

	<pre>End If End Sub</pre>
See Also	Variant (on page 771) (data type); IsEmpty (on page 572) (function); IsError (on page 572) (function); IsObject (on page 575) (function); VarType (on page 773) (function); IsNull (on page 574) (function).

IsEmpty (function)

Syntax	IsEmpty (expression)
Description	Returns True if expression is a Variant variable that has never been initialized; returns False otherwise.
Comments	The IsEmpty function is the same as the following: <pre>(VarType(expression) = ebEmpty)</pre>
Example	<pre>Sub Main() Dim a As Variant If IsEmpty(a) Then a = 1.0# 'Give uninitialized data a Double value 0.0. MsgBox "The variable has been initialized to: " & a Else MsgBox "The variable was already initialized!" End If End Sub</pre>
See Also	Variant (on page 771) (data type); IsDate (on page 571) (function); IsError (on page 572) (function); IsObject (on page 575) (function); VarType (on page 773) (function); IsNull (on page 574) (function).

IsError (function)

Syntax	IsError (expression)
Description	Returns True if expression is a user-defined error value; returns False otherwise.

Ex-ample	<p>This example creates a function that divides two numbers. If there is an error dividing the numbers, then a variant of type "error" is returned. Otherwise, the function returns the result of the division. The <code>IsError</code> function is used to determine whether the function encountered an error.</p> <pre> Function Div(ByVal a,ByVal b) As Variant If b = 0 Then Div = CVErr(2112) 'Return a special error value. Else Div = a / b 'Return the division. End If End Function Sub Main() Dim a As Variant a = Div(10,12) If IsError(a) Then MsgBox "The following error occurred: " & CStr(a) Else MsgBox "The result of the division is: " & a End If End Sub </pre>
See Also	<p>Variant (on page 771) (data type); IsEmpty (on page 572) (function); IsDate (on page 571) (function); IsObject (on page 575) (function); VarType (on page 773) (function); IsNull (on page 574) (function).</p>

IsMissing (function)

Syntax	IsMissing (variable)
Description	Returns True if variable was passed to the current subroutine or function; returns False if omitted.
Comments	The IsMissing is used with variant variables passed as optional parameters (using the Optional keyword) to the current subroutine or function. For non-variant variables or variables that were not declared with the Optional keyword, IsMissing will always return True .
Example	The following function runs an application and optionally minimizes it. If the optional <code>isMinimize</code> parameter is not specified by the caller, then the application is not minimized.

	<pre> Sub Test(AppName As String,Optional isMinimize As Variant) app = Shell(AppName) If Not IsMissing(isMinimize) Then AppMinimize app Else AppMaximize app End If End Sub Sub Main Test "notepad.exe" 'Maximize this application Test "notepad.exe",True 'Minimize this application End Sub </pre>
See Also	Declare (on page 412) (statement), Sub...End Sub (on page 744) (statement), Function...End Function (statement) (on page 533)

IsNull (function)

Syntax	IsNull (expression)
Description	Returns True if expression is a Variant variable that contains no valid data; returns False otherwise.
Comments	<p>The IsNull function is the same as the following:</p> <pre>(VarType(expression) = vbNull)</pre>
Example	<pre> Sub Main() Dim a As Variant 'Initialized as Empty If IsNull(a) Then MsgBox "The variable contains no valid data." a = Empty * Null If IsNull(a) Then MsgBox "Null propagated through the expression." End Sub </pre>
See Also	Empty (on page 486) (constant); Variant (on page 771) (data type); IsEmpty (on page 572) (function); IsDate (on page 571) (function); IsError (on page 572) (function); IsObject (on page 575) (function); VarType (on page 773) (function).

IsNumeric (function)

Syntax	IsNumeric (expression)
Description	Returns True if expression can be converted to a number; returns False otherwise.
Comments	<p>If passed a number or a variant containing a number, then IsNumeric always returns True . If a String or String variant is passed, then IsNumeric will return True only if the string can be converted to a number. The following syntaxes are recognized as valid numbers:</p> <pre>&Hhexdigits[& % ! # @] &[O]octaldigits[& % ! # @] [- +]digits[.[digits]][E[- +]digits][! % & # @]</pre> <p>If an Object variant is passed, then the default property of that object is retrieved and one of the above rules is applied. IsNumeric returns False if expression is a Date .</p>
Example	<pre>Sub Main() Dim s\$ As String s\$ = InputBox("Enter a number.", "Enter Number") If IsNumeric(s\$) Then MsgBox "You did good!" Else MsgBox "You didn't do so good!" End If End Sub</pre>
See Also	Variant (on page 771) (data type); IsEmpty (on page 572) (function); IsDate (on page 571) (function); IsError (on page 572) (function); IsObject (on page 575) (function); VarType (on page 773) (function); IsNull (on page 574) (function).

IsObject (function)

Syntax	IsObject (expression)
Description	Returns True if expression is a Variant variable containing an Object ; returns False otherwise.

<p>Example</p>	<p>This example will attempt to find a running copy of Excel and create 'a Excel object that can be referenced as any other object in the Basic Control Engine.</p> <pre style="background-color: #f0f0f0; padding: 10px;"> Sub Main() Dim v As Variant On Error Resume Next Set v = GetObject(,"Excel.Application") If IsObject(v) Then MsgBox "The default object value is: " & v = v.Value 'Access value property of the object. Else MsgBox "Excel not loaded." End If End Sub </pre>
<p>See Also</p>	<p>Variant (on page 771) (data type); IsEmpty (on page 572) (function); IsDate (on page 571) (function); IsError (on page 572) (function); VarType (on page 773) (function); IsNull (on page 574) (function).</p>

IsWebSpaceSession (function)

Syntax IsWebSpaceSession

Description Returns True if CimView is opened in Webspace session..

Example Sub Main()

 MsgBox "WebSpace Session = " & IsWebSpaceSession

 End Sub

Item\$ (function)

<p>Syntax</p>	<p>Item\$ (text\$,first,last [,delimiters\$])</p>
<p>Description</p>	<p>Returns all the items between first and last within the specified formatted text list.</p>

Comments	The Item\$ function takes the following parameters:	
	Parameter	Description
	text\$	String containing the text from which a range of items is returned.
	first	Integer containing the index of the first item to be returned. If first is greater than the number of items in text\$, then a zero-length string is returned.
	last	Integer containing the index of the last item to be returned. All of the items between first and last are returned. If last is greater than the number of items in text\$, then all items from first to the end of text are returned.
	delimiters\$	String containing different item delimiters. By default, items are separated by commas and end-of-lines. This can be changed by specifying different delimiters in the delimiters\$ parameter.
Example	<p>This example creates two delimited lists and extracts a range from each, then displays the result in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() ildlist\$ = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15" slist\$ = "1/2/3/4/5/6/7/8/9/10/11/12/13/14/15" list1\$ = Item\$(ildlist\$,5,12) list2\$ = Item\$(slist\$,2,9,"/") MsgBox "The returned lists are: " & crlf & list1\$ & crlf & list2\$ End Sub </pre>	
See Also	ItemCount (on page 577) (function); Line\$ (on page 589) (function); LineCount (on page 590) (function); Word\$ (on page 792) (function); WordCount (on page 793) (function).	

ItemCount (function)

Syntax	ItemCount (text\$ [,delimiters\$])
--------	---

De- scrip- tion	Returns an Integer containing the number of items in the specified delimited text.
Com- ments	Items are substrings of a delimited text string. Items, by default, are separated by commas and/or end-of-lines. This can be changed by specifying different delimiters in the delimiters\$ parameter. For example, to parse items using a backslash: <pre>n = ItemCount(text\$, "\")</pre>
Exam- ple	This example creates two delimited lists and then counts the number of items in each. The counts are displayed in a dialog box. <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() ildist\$ = "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15" slist\$ = "1/2/3/4/5/6/7/8/9/10/11/12/13/14/15/16/17/18/19" l1% = ItemCount(ildist\$) l2% = ItemCount(slist\$, "/") msg1 = "The first lists contains: " & l1% & " items." & crlf msg1 = msg1 & "The second list contains: " & l2% & " items." MsgBox msg1 End Sub</pre>
See Also	Item\$ (on page 576) (function); Line\$ (on page 589) (function); LineCount (on page 590) (function); Word\$ (on page 792) (function); WordCount (on page 793) (function).

K

K

Keywords (top- ic)
Kill (statement)

Keywords (topic)

A keyword is any word or symbol recognized by the Basic Control Engine as part of the language. All of the following are keywords:

- Built-in subroutine names, such as **MsgBox** and **Print**.
- Built-in function names, such as **Str\$**, **Cdbl**, and **Mid\$**.
- Special keywords, such as **To**, **Next**, **Case**, and **Binary**.
- Names of any extended language elements.

Restrictions All keywords are reserved by the Basic Control Engine , in that you cannot create a variable, function, constant, or subroutine with the same name as a keyword. However, you are free to use all keywords as the names of structure members.

Kill (statement)

Syntax	Kill filespec\$		
Description	Deletes all files matching filespec\$.		
Comments	<p>The filespec\$ argument can include wildcards, such as * and ?. The * character matches any sequence of zero or more characters, whereas the ? character matches any single character. Multiple * 's and ? 's can appear within the expression to form complex searching patterns. The following table shows some examples.</p>		
	This Pattern	Matches these Files	Doesn't match these Files
	S.TXT	SAMPLE.TXT GOOSE.TXT SAMS.TXT	SAMPLE SAMPLE.DAT
	C*T.TXT	CAT.TXT	CAP.TXT ACATS.TXT
	C*T	CAT	CAT.DOC CAP.TXT
	C?T	CAT CUT	CAT.TXT CAPIT CT
	*	(All files)	
Example	<p>This example looks to see whether file test1.dat exists. If it does not, then it creates both test1.dat and test2.dat. The existence of the files is tested again; if they exist, a message is generated, and then they are deleted. The final test looks to see whether they are still there and displays the result.</p> <pre> Sub Main() If Not FileExists("test1.dat") Then Open "test1.dat" For Output As #1 Open "test2.dat" For Output As #2 </pre>		

	<pre> Close End If If FileExists ("test1.dat") Then MsgBox "File test1.dat exists." Kill "test?.dat" End If If FileExists ("test1.dat") Then MsgBox "File test1.dat still exists." Else MsgBox "test?.dat successfully deleted." End If End Sub </pre>
<p>See Also</p>	<p>Name (on page 619) (statement).</p>

L

L

LBound (function)
LCase, LCase\$ (function)
Left, Left\$, LeftB, LeftB\$ (functions)
Len (function)
Let (statement)
Like (operator)
Line Input# (statement)
Line Numbers (topic)
Line\$ (function)
LineCount (function)
ListBox (statement)
Literals (topic)
Loc (function)

Lock (statement)
Lof (function)
Log (function)
Long (data type)
LSet (statement)
LTrim, LTrim\$ (functions)

LBound (function)

Syn- tax	LBound (ArrayVariable() [,dimension])
De- scrip- tion	Returns an Integer containing the lower bound of the specified dimension of the specified array variable.
Com- ments	<p>The dimension parameter is an integer specifying the desired dimension. If this parameter is not specified, then the lower bound of the first dimension is returned. The LBound function can be used to find the lower bound of a dimension of an array returned by an OLE automation method or property:</p> <pre>LBound(object.property [,dimension]) LBound(object.method [,dimension])</pre>
Exam- ples	<pre>Sub Main() 'This example dimensions two arrays and displays their lower bounds. Dim a(5 To 12) Dim b(2 To 100,9 To 20) lba = LBound(a) lbb = LBound(b,2) MsgBox "The lower bound of a is: " & lba & " The lower bound of b is: " & lbb</pre>
	<pre>'This example uses LBound and UBound to dimension a dynamic array to 'hold a copy of an array redimmed by the FileList statement. Dim fl\$() FileList fl\$,"*.*" count = UBound(fl\$) If ArrayDims(a) Then</pre>

	<pre> Redim nl\$(LBound(fl\$) To UBound(fl\$)) For x = 1 To count nl\$(x) = fl\$(x) Next x MsgBox "The last element of the new array is: " & nl\$(count) End If End Sub </pre>
See Also	UBound (on page 765) (function); ArrayDims (on page 328) (function); Arrays (on page 329) (topic).

LCase, LCase\$(functions)

Syntax	LCase[\$](text)
Description	Returns the lowercase equivalent of the specified string.
Comments	LCase\$ returns a String , whereas LCase returns a String variant. Null is returned if text is Null .
Example	<p>This example shows the LCase function used to change uppercase names to lowercase with an uppercase first letter.</p> <pre> Sub Main() lname\$ = "WILLIAMS" fl\$ = Left(lname\$,1) rest\$ = Mid(lname\$,2,Len(lname\$)) lname\$ = fl\$ & LCase(rest\$) MsgBox "The converted name is: " & lname\$ End Sub </pre>
See Also	UCase (on page 766) , UCase\$ (on page 766) (functions).

Left, Left\$, LeftB, LeftB\$(functions)

Syntax	Left[\$](string, length) LeftB[\$](string,length)
--------	---

De- scrip- tion	Functions return the leftmost length characters as follows.		
	Functions	Return the leftmost length characters	
	Left and Left\$	Of bytes	
	LeftB and LeftB\$	From a given string	
	Left\$ and Left functions return the following.		
	Function	Returns	
	Left\$	String	
	Left	String variant.	
		The length parameter is an Integer value specifying the number of characters to return as follows.	
		Length is	Returns
		0	Zero-length string
		Greater than or equal to the number of characters in the specified string	Entire string
	LeftB and LeftB\$ functions return the following.		
	Functions	Return	
	LeftB and LeftB\$	Sequence of bytes from a string containing byte data.	
		length specifies the number of bytes to return as follows.	
		Length is	Returns
		Greater than the number of bytes in string	Entire string
		String is Null.	Null
Com- ments	Left\$ returns a String , whereas Left returns a String variant. NumChars is an Integer value specifying the number of character to return. If NumChars is 0, then a zero-length string is returned. If NumChars is greater than or equal to the number of characters in the specified string, then the entire string is returned. NULL is returned if text is NULL.		

Example	<p>This example shows the Left\$ function used to change uppercase names to lowercase with an uppercase first letter.</p> <pre data-bbox="305 302 1408 636"> Sub Main() lname\$ = "WILLIAMS" fl\$ = Left(lname\$,1) rest\$ = Mid(lname\$,2,Len(lname\$)) lname\$ = fl\$ & LCase(rest\$) MsgBox "The converted name is: " & lname\$ End Sub </pre>
See Also	Right , Right\$, RightB , RightB\$ (<i>on page 692</i>) (functions)

Len (function)

Syntax	Len (expression)	
Description	Returns the number of characters in expression or the number of bytes required to store the specified variable.	
Comments	<p>If expression evaluates to a string, then Len returns the number of characters in a given string or 0 if the string is empty. When used with a Variant variable, the length of the variant when converted to a String is returned. If expression is a Null, then Len returns a Null variant. If used with a non-String or non-Variant variable, the function returns the number of bytes occupied by that data element. When used with user-defined data types, the function returns the combined size of each member within the structure. Since variable-length strings are stored elsewhere, the size of each variable-length string within a structure is 2 bytes. The following table describes the sizes of the individual data elements:</p>	
	Data Element	Size
	Integer	2 bytes
	Long	4 bytes
	Float	4 bytes

	Dou- ble	8 bytes.
	Cur- rency	8 bytes
	String (vari- able-length)	Number of characters in the string.
	String (fixed- length)	The length of the string as it appears in the string's declaration.
	Ob- jects	0 bytes. Both data object variables and variables of type Object are always returned as 0 size.
	User- de- fined type	Combined size of each structure member. Variable-length strings within structures require 2 bytes of storage. Arrays within structures are fixed in their dimensions. The elements for fixed arrays are stored within the structure and therefore require the number of bytes for each array element multiplied by the size of each array dimension: <code>element_size * dimension1 * dimension2...</code>
The Len function always returns 0 with object variables or any data object variable.		
Exam- ple	<p>Const crlf = Chr\$(13) + Chr\$(10)</p> <pre> Sub Main() 'This example shows the Len function used in a routine to change 'uppercase names to lowercase with an uppercase first letter. lname\$ = "WILLIAMS" fl\$ = Left(lname\$,1) ln% = Len(lname\$) rest\$ = Mid(lname\$,2,ln%) nname\$ = fl\$ & LCase(rest\$) MsgBox "The proper case for " & lname\$ & " is " & nname\$ & "." </pre>	
	<pre> 'This example returns a table of lengths for standard numeric types. Dim lns(4) a% = 100 : b& = 200 : c! = 200.22 : d# = 300.22 lns(1) = Len(a%) lns(2) = Len(b&) </pre>	

	<pre> lns(3) = Len(c!) lns(4) = Len(d#) msg1 = "Lengths (in bytes) of standard types:" & crlf & crlf msg1 = msg1 & "Integer: " & lns(1) & crlf msg1 = msg1 & "Long: " & lns(2) & crlf msg1 = msg1 & "Single: " & lns(3) & crlf msg1 = msg1 & "Double: " & lns(4) & crlf MsgBox msg1 End Sub </pre>
See Also	InStr (on page 563) (function)

Let (statement)

Syn- tax	[Let] variable = expression
De- scrip- tion	Assigns the result of an expression to a variable.
Com- ments	<p>The use of the word Let is supported for compatibility with other implementations of the Basic Control Engine. Normally, this word is dropped. When assigning expressions to variables, internal type conversions are performed automatically between any two numeric quantities. Thus, you can freely assign numeric quantities without regard to type conversions. However, it is possible for an overflow error to occur when converting from larger to smaller types. This happens when the larger type contains a numeric quantity that cannot be represented by the smaller type. For example, the following code will produce a runtime error:</p> <pre> Dim amount As Long im quantity As Integer amount = 400123 'Assign a value out of range for int. quantity = amount 'Attempt to assign to Integer. </pre> <p>When performing an automatic data conversion, underflow is not an error.</p>
Exam- ple	<pre> Sub Main() Let a\$ = "This is a string." Let b% = 100 </pre>

	<pre>Let c# = 1213.3443 End Sub</pre>
See Also	= (on page 305) (keyword); Expression Evaluation (on page 509) (topic).

Like (operator)

Syntax	expression Like pattern		
Description	Compares two strings and returns TRUE if the expression matches the given pattern; returns FALSE otherwise.		
Comments	Case sensitivity is controlled by the Option Compare setting. The pattern expression can contain special characters that allow more flexible matching:		
	Character	Evaluates to	
	?	Matches a single character.	
	*	Matches one or more characters.	
	#	Matches any digit.	
	[range]	Matches if the character in question is within the specified range.	
	[!range]	Matches if the character in question is not within the specified range.	
	A range specifies a grouping of characters. To specify a match of any of a group of characters, use the syntax [ABCDE] . To specify a range of characters, use the syntax [A-Z] . Special characters must appear within brackets, such as [*?#] . If expression or pattern is not a string, then both expression and pattern are converted to String variants and compared, returning a Boolean variant. If either variant is Null , then Null is returned. The following table shows some examples:		
	expression	TRUE If pattern Is	FALSE If pattern is Is
	"EBW"	"E*W", "E*"	"E*B"
	"BasicScript"	"B*[r-t]icScript"	"B[r-t]ic"
	"Version"	"V[e]?s*n"	"V[r]?s*N"
	"2.0"	"#.##", "#?#"	"###", "#?[!0-9]"

	"[ABC]"	"[[]*]"	"[ABC]", "[*]"
Example	<p>This example demonstrates various uses of the Like function.</p> <pre> Sub Main() a\$ = "This is a string variable of 123456 characters" b\$ = "123.45" If a\$ Like "[A-Z][g-i]*" Then MsgBox "The first comparison is True." If b\$ Like "##3.##" Then MsgBox "The second comparison is True." If a\$ Like "**variable*" Then MsgBox "The third comparison is True." End Sub </pre>		
See Also	Operator Precedence (on page 648) (topic); Is (on page 570) (operator); Option Compare (on page 649) (statement).		

Line Input# (statement)

Syntax	Line Input [#] filename,variable
Description	Reads an entire line into the given variable.
Comments	<p>The filename parameter is a number that is used to refer to the open file; the number passed to the Open statement. The filename must reference a file opened in Input mode. The file is read up to the next end-of-line, but the end-of-line character(s) is (are) not returned in the string. The file pointer is positioned after the terminating end-of-line.</p> <p>The variable parameter is any string or variant variable reference. This statement will automatically declare the variable if the specified variable has not yet been used or dimensioned. This statement recognizes either a single line feed or a carriage-return/line-feed pair as the end-of-line delimiter.</p>
Example	<p>This example reads five lines of the autoexec.bat file and displays them in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() file\$ = "c:\autoexec.bat" Open file\$ For Input As #1 msg1 = "" For x = 1 To 5 </pre>

	<pre> Line Input #1,lin\$ msg1 = msg1 & lin\$ & crlf Next x MsgBox "The first 5 lines of '" & file\$ & "' are:" & crlf & crlf & msg1 End Sub </pre>
See	Open (on page 642) (statement); Get (on page 535) (statement); Input# (on page 559)
Also	(statement); Input, Input\$ (on page 560) (functions).

Line\$ (function)

Syn- tax	Line\$(text\$,first[,last])								
De- scrip- tion	Returns a String containing a single line or a group of lines between first and last.								
Com- ments	Lines are delimited by carriage return, line feed, or carriage-return/line-feed pairs. The <code>Line\$</code> function takes the following parameters:								
	<table border="1"> <thead> <tr> <th>Pa- ra- me- ter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>text \$</td> <td>String containing the text from which the lines will be extracted.</td> </tr> <tr> <td>first</td> <td>Integer representing the index of the first line to return. If last is omitted, then this line will be returned. If first is greater than the number of lines in text\$, then a zero-length string is returned.</td> </tr> <tr> <td>last</td> <td>Integer representing the index of the last line to return.</td> </tr> </tbody> </table>	Pa- ra- me- ter	Description	text \$	String containing the text from which the lines will be extracted.	first	Integer representing the index of the first line to return. If last is omitted, then this line will be returned. If first is greater than the number of lines in text\$, then a zero-length string is returned.	last	Integer representing the index of the last line to return.
Pa- ra- me- ter	Description								
text \$	String containing the text from which the lines will be extracted.								
first	Integer representing the index of the first line to return. If last is omitted, then this line will be returned. If first is greater than the number of lines in text\$, then a zero-length string is returned.								
last	Integer representing the index of the last line to return.								
Exam- ple	<p>This example reads five lines of the autoexec.bat file, extracts the third and fourth lines with the Line\$ function, and displays them in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() file\$ = "c:\autoexec.bat" Open file\$ For Input As #1 txt = "" </pre>								

	<pre> For x = 1 To 5 Line Input #1,lin\$ txt = txt & lin\$ & crlf Next x lines\$ = Line\$(txt,3,4) MsgBox "The 3rd and 4th lines of '" & file\$ & "' are:" & crlf_ & crlf & lines\$ End Sub </pre>
See Also	Item\$ (on page 576) (function); ItemCount (on page 577) (function); LineCount (on page 590) (function); Word\$ (on page 792) (function); WordCount (on page 793) (function).

LineCount (function)

Syntax	LineCount (text\$)
Description	Returns an Integer representing the number of lines in text\$.
Comments	Lines are delimited by carriage return, line feed, or both.
Example	<p>This example reads your autoexec.bat file into a variable and then determines how many lines it is comprised of.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() file\$ = "c:\autoexec.bat" Open file\$ For Input As #1 txt = "" Do Until Eof(1) Line Input #1,lin\$ txt = txt & lin\$ & crlf Loop lines! = LineCount(txt) MsgBox "" & file\$ & " is " & lines! & " lines long!" & crlf_ & crlf & txt End Sub </pre>

See Also [Item\\$ \(on page 576\)](#) (function); [ItemCount \(on page 577\)](#) (function); [Line\\$ \(on page 589\)](#) (function); [Word\\$ \(on page 792\)](#) (function); [WordCount \(on page 793\)](#) (function).

Line Numbers (topic)

Line numbers are not supported by the Basic Control Engine. As an alternative to line numbers, you can use meaningful labels as targets for absolute jumps, as shown below:

```
Sub Main()
    Dim i As Integer
    On Error Goto MyErrorTrap
    i = 0
LoopTop:
    i = i + 1
    If i < 10 Then Goto LoopTop
MyErrorTrap:
    MsgBox "An error occurred."
End Sub
```

ListBox (statement)

Syn- tax	ListBox X,Y,width,height,ArrayVariable,.Identifier	
De- scrip- tion	Creates a list box within a dialog box template.	
Com- ments	When the dialog box is invoked, the list box will be filled with the elements contained in Array-Variable. This statement can only appear within a dialog box template (that is, between the Be- gin Dialog and End Dialog statements). The ListBox statement requires the following parameters:	
	Para- meter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.

	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Array- Vari- able	Specifies a single-dimensioned array of strings used to initialize the elements of the list box. If this array has no dimensions, then the list box will be initialized with no elements. A runtime error results if the specified array contains more than one dimension. Array-Variable can specify an array of any fundamental data type (structures are not allowed). Null and Empty values are treated as zero-length strings.
	Identi- fier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the list box's selection (0 is the first item, 1 is the second, and so on). This variable can be accessed using the following syntax: DialogVariable . Identifier
Exam- ple	<p>This example creates a dialog box with two list boxes, one containing files and the other containing directories.</p> <pre> Sub Main() Dim files() As String Dim dirs() As String Begin Dialog ListBoxTemplate 16,32,184,96,"Sample" Text 8,4,24,8,"&Files:" ListBox 8,16,60,72,files\$,.Files Text 76,4,21,8,"&Dirs:" ListBox 76,16,56,72,dirs\$,.Dirs OKButton 140,4,40,14 CancelButton 140,24,40,14 End Dialog FileList files FileDirs dirs Dim ListBoxDialog As ListBoxTemplate rc% = Dialog(ListBoxDialog) End Sub </pre>	
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); OK-Button (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup	

([on page 653](#)) (statement); [Picture \(on page 657\)](#) (statement); [PushButton \(on page 659\)](#) (statement); [Text \(on page 752\)](#) (statement); [TextBox \(on page 753\)](#) (statement); [Begin \(on page 348\)](#) [Dialog \(on page 348\)](#) (statement), [PictureBox \(on page 659\)](#) (statement).

Literals (topic)

Literals are values of a specific type. The following table shows the different types of literals supported by the Basic Control Engine:

Literal	Description	
10	Integer whose value is 10.	
43265	Long whose value is 43,265.	
5#	Double whose value is 5.0. A number's type can be explicitly set using any of the following type-declaration characters:	
	%	Integer
	&	Long
	#	Double
	!	Single
5.5	Double	Value is 5.5. Any number with decimal point is considered a double.
5.4E100	Double	Expressed in scientific notation.
&HFF	Integer	Expressed in hexadecimal.
&O47	Integer	Expressed in octal.
&HFF#	Double	Expressed in hexadecimal.
"hello"	String	Of five characters: hello .
""hello""	String	Of seven characters: "hello" . Quotation marks can be embedded within strings by using two consecutive quotation marks.

#1/1/1994#	<p>Date value whose internal representation is 34335.0. Any valid date can appear with #'s. Date literals are interpreted at execution time using the locale settings of the host environment. To ensure that date literals are correctly interpreted for all locales, use the international date format: #YYYY-MM-DD HH:MM:SS#</p> <p>Constant Folding The Basic Control Engine supports constant folding where constant expressions are calculated by the compiler at compile time. For example, the expression <code>i% = 10 + 12</code> is the same as: <code>i% = 22</code> Similarly, with strings, the expression <code>s\$ = "Hello," + " there" + (46)</code> is the same as: <code>s\$ = "Hello, there."</code></p>
------------	---

Loc (function)

Syntax	Loc (filenumber)	
Description	Returns a Long representing the position of the file pointer in the given file.	
Comments	The filenumber parameter is an Integer used by the Basic Control Engine to refer to the number passed by the Open statement to the Basic Control Engine . The Loc function returns different values depending on the mode in which the file was opened:	
	File Mode	Returns
	Input	Current byte position divided by 128.
	Output	Current byte position divided by 128.
	Append	Current byte position divided by 128.
	Binary	Position of the last byte read or written.
	Random	Number of the last record read or written.
Example	<p>This example reads 5 lines of the autoexec.bat file, determines the current location of the file pointer, and displays it in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() file\$ = "c:\autoexec.bat" Open file\$ For Input As #1 For x = 1 To 5 If Not EOF(1) Then Line Input #1,lin\$ </pre>	

	<pre> Next x lc% = Loc(1) Close MsgBox "The file byte location is: " & lc% End Sub </pre>
See	Seek (on page 706) (function); Seek (on page 707) (statement); FileLen (on page 517)
Also	(function).

Lock (statement)

Syn- tax	<code>Lock [#] filename [{record [start] To end}]</code>	
De- scrip- tion	Locks a section of the specified file, preventing other processes from accessing that section of the file until the Unlock statement is issued.	
Com- ments	The <code>Lock</code> statement requires the following parameters:	
	Parameter	Description
	filename	Integer used by the Basic Control Engine to refer to the open file—the number passed to the Open statement.
	record	Long specifying which record to lock.
	start	Long specifying the first record within a range to be locked.
	end	Long specifying the last record within a range to be locked.
	For sequential files, the record, start, and end parameters are ignored. The entire file is locked. The section of the file is specified using one of the following:	
	Syntax	Description
	No para- meters	Locks the entire file (no record specification is given).
	record	Locks the specified record number (for Random files) or byte (for Binary files).
	to end	Locks from the beginning of the file to the specified record (for Random files) or byte (for Binary files).

	start to end	Locks the specified range of records (for Random files) or bytes (for Binary files).
	<p>The lock range must be the same as that used to subsequently unlock the file range, and all locked ranges must be unlocked before the file is closed. Ranges within files are not unlocked automatically by the Basic Control Engine when your script terminates, which can cause file access problems for other processes. It is a good idea to group the Lock and Unlock statements close together in the code, both for readability and so subsequent readers can see that the lock and unlock are performed on the same range. This practice also reduces errors in file locks.</p>	
Example	<p>This example creates test.dat and fills it with ten string variable records. These are displayed in a dialog box. The file is then reopened for read/write, and each record is locked, modified, rewritten, and unlocked. The new records are then displayed in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = "This is record number: " b\$ = "0" rec\$ = "" msg1 = "" Open "test.dat" For Random Access Write Shared As #1 For x = 1 To 10 rec\$ = a\$ & x Lock #1,x Put #1,,rec\$ Unlock #1,x msg1 = msg1 & rec\$ & crlf Next x Close MsgBox "The records are:" & crlf & msg1 msg1 = "" Open "test.dat" For Random Access Read Write Shared As #1 For x = 1 To 10 rec\$ = Mid(rec\$,1,23) & (11 - x) Lock #1,x Put #1,x,rec\$ Unlock #1,x msg1 = msg1 & rec\$ & crlf </pre>	

	<pre> Next x MsgBox "The records are: " & crlf & msg1 Close Kill "test.dat" End Sub </pre>
See Also	Unlock (on page 766) (statement); Open (on page 642) (statement).

Lof (function)

Syntax	Lof (filenumber)
Description	Returns a Long representing the number of bytes in the given file.
Comments	The filenumber parameter is an Integer used by the Basic Control Engine to refer to the open file, the number passed to the Open statement. The file must currently be open.
Example	<p>This example creates a test file, writes ten records into it, then finds the length of the file and displays it in a message box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = "This is record number: " Open "test.dat" For Random Access Write Shared As #1 msg1 = "" For x = 1 To 10 rec\$ = a\$ & x put #1,,rec\$ msg1 = msg1 & rec\$ & crlf Next x Close Open "test.dat" For Random Access Read Write Shared As #1 r% = Lof(1) Close </pre>

	<pre>MsgBox "The length of 'test.dat' is: " & r% End Sub</pre>
See Also	Loc (on page 594) (function); Open (on page 642) (statement); FileLen (on page 517) (function).

Log (function)

Syntax	Log (number)
Description	Returns a Double representing the natural logarithm of a given number.
Comments	The value of number must be a Double greater than 0. The value of e is 2.71828.
Example	<p>This example calculates the natural log of 100 and displays it in a message box.</p> <pre>Sub Main() x# = Log(100) MsgBox "The natural logarithm of 100 is: " & x# End Sub</pre>
See Also	Exp (on page 508) (function).

Long (data type)

Syntax	Long
Description	<p>Long variables are used to hold numbers (with up to ten digits of precision) within the following range:</p> <pre>-2,147,483,648 <= Long <= 2,147,483,647</pre> <p>Internally, longs are 4-byte values. Thus, when appearing within a structure, longs require 4 bytes of storage. When used with binary or random files, 4 bytes of storage are required. The type-declaration character for Long is &.</p>
See Also	Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Integer (on page 566) (data type); Object (on page 633) (data type); Single (on page 718) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement); CLng (on page 372) (function).

LSet (statement)

Syntax 1	<pre>LSet dest = source</pre>
Syntax 2	<pre>LSet dest_variable = source_variable</pre>
Description	<p>Left-aligns the source string in the destination string or copies one user-defined type to another.</p>
Comments	<p>Syntax 1 The LSet statement copies the source string source into the destination string dest. The dest parameter must be the name of either a String or Variant variable. The source parameter is any expression convertible to a string. If source is shorter in length than dest, then the string is left-aligned within dest, and the remaining characters are padded with spaces. If source \$ is longer in length than dest, then source is truncated, copying only the leftmost number of characters that will fit in dest. The destvariable parameter specifies a String or Variant variable. If destvariable is a Variant containing Empty, then no characters are copied. If destvariable is not convertible to a String, then a runtime error occurs. A runtime error results if destvariable is Null. Syntax 2 The source structure is copied byte for byte into the destination structure. This is useful for copying structures of different types. Only the number of bytes of the smaller of the two structures is copied. Neither the source structure nor the destination structure can contain strings.</p>
Example	<p>This example replaces a 40-character string of asterisks (*) with an RSet and LSet string and then displays the result.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim msg,tmpstr\$ tmpstr\$ = String(40,"*") msg1 = "Here are two strings that have been right-" + crlf msg1 = msg1 & "and left-justified in a 40-character string." Msg1 = msg1 & crlf & crlf Rset tmpstr\$ = "Right " msg1 = msg1 & tmpstr\$ & crlf LSet tmpstr\$ = " Left" msg1 = msg1 & tmpstr\$ & crlf</pre>

	<pre>MsgBox msg1 End Sub</pre>
See Also	RSet (on page 695) (function).

LTrim, LTrim\$ (functions)

Syntax	<code>LTrim[\$](text)</code>
Description	Returns text with the leading spaces removed.
Comments	LTrim\$ returns a String , whereas LTrim returns a String variant. Null is returned if text is Null .
Example	<p>This example displays a right-justified string and its LTrim result. Const crlf = Chr\$(13) + Chr\$(10)</p> <pre>Sub Main() txt\$ = " This is text " tr\$ = LTrim(txt\$) MsgBox "Original ->" & txt\$ & "<->" & crlf & "Left Trimmed ->" & tr\$ & "<->" End Sub</pre>
See Also	RTrim , RTrim\$ (on page 696) (functions); Trim , Trim\$ (on page 758) (functions).

M

M

Main (statement)
MCI (function)
Mid, Mid\$, MidB, MidB\$ (functions)
Mid, Mid\$, MidB, MidB\$ (statements)
Minute (function)
MIRR (function)
MkDir (statement)

Mod (operator)
Month (function)
Msg.Close (method)
Msg.Open (method)
Msg.Text (property)
Msg.Thermometer (property)
MsgBox (function)
MsgBox (statement)

Main (statement)

Syntax	<pre>Sub Main() End Sub</pre>
Description	Defines the subroutine where execution begins.
Example	<pre>Sub Main() MsgBox "This is the Main() subroutine and entry point." End Sub</pre>

Mci (function)

Syntax	Mci (command\$,result\$ [,error\$])	
Description	Executes an Mci command, returning an Integer indicating whether the command was successful.	
Comments	The Mci function takes the following parameters:	
	Parameter	Description

	com- mand\$	String containing the command to be executed.
	re- sult\$	String variable into which the result is placed. If the command doesn't return anything, then a zero-length string is returned. To ignore the returned string, pass a zero-length string, such as. r% = Mci("open chimes.wav type waveaudio","")
	er- ror\$	Optional String variable into which an error string will be placed. A zero-length string will be returned if the function is successful.
Exam- ple 1	<p>This first example plays a wave file. The wave file is played to completion before execution can continue.</p> <pre> Sub Main() Dim result As String Dim ErrorMessage As String Dim Filename As String Dim rc As Integer 'Establish name of file in the Windows directory. Filename = FileParse\$(System.WindowsDirectory\$ + "\" + "chimes.wav") 'Open the file and driver. rc = Mci("open " & Filename & " type waveaudio alias CoolSound","",ErrorMessage) If (rc) Then 'Error occurred--display error message to user. MsgBox ErrorMessage Exit Sub End If rc = Mci("play CoolSound wait","",") 'Wait for sound to finish. rc = Mci("close CoolSound","",") 'Close driver and file. End Sub </pre>	
Exam- ple 2	<p>This next example shows how to query an Mci device and play an MIDI file in the background.</p> <pre> Sub Main() Dim result As String Dim ErrMsg As String Dim Filename As String Dim rc As Integer 'Check to see whether MIDI device can play for us. rc = Mci("capability sequencer can play",result,ErrorMessage) 'Check for error. </pre>	

	<pre> If rc Then MsgBox ErrorMessage Exit Sub End If 'Can it play? If result <> "true" Then MsgBox "MIDI device is not capable of playing." Exit Sub End If 'Assemble a filename from the Windows directory. Filename = FileParse\$(System.WindowsDirectory\$ & "\\\" & "canyon.mid") 'Open the driver and file. rc = Mci("open " & Filename & " type sequencer alias song",result\$,ErrMsg) If rc Then MsgBox ErrMsg Exit Sub End If rc = Mci("play song","","") 'Play in the background. MsgBox "Press OK to stop the music.",ebOKOnly rc = Mci("close song","","") End Sub </pre>
See Also	Beep (on page 348) (statement)

Mid, Mid\$, MidB, MidB\$ (functions)

Syntax	<code>Mid[\$](string, start [,length])</code> <code>MidB[\$](string, start [,length])</code>
Description	Returns a sub-string of the specified string, beginning with start, for length characters (for <code>Mid</code> and <code>Mid\$</code>) or bytes (for <code>MidB</code> and <code>MidB\$</code>).
Comments	The functions start and length are:

	Func- tions	Start and Length of Return
	Mid and Mid\$	Sub-string starting at character position start and will be length characters long
	MidB and MidB\$	Sub-string starting at byte position start and will be length bytes long.
The functions return the following.		
	Func- tions	Return
	Mid\$ and MidB	String
	Mid and MidB	String variant
The returned sub-string starts at character position start and will be length characters long. Mid \$ returns a String , whereas Mid returns a String variant. The Mid/Mid\$ functions take the following parameters:		
	Parame- ter	Description
	string	Any String expression containing the text from which data is returned.
	start	Integer specifying the position where the sub-string begins. If start is greater than the length of string, then a zero-length string is returned.
	length	Integer specifying the number of characters or bytes to return. If this parameter is omitted, then the entire string is returned, starting at start.
The Mid function will return Null text is Null . The MidB and MidB\$ functions are used to return a sub-string of bytes from a string containing byte data.		
Exam- ple	<pre> This example displays a substring from the middle of a string variable using the Mid\$ function and replaces the first four characters with "NEW " using the Mid\$ statement. Const crlf = Chr\$(13) + Chr\$(10) </pre>	

	<pre> Sub Main() a\$ = "This is the Main string containing text." b\$ = Mid\$(a\$,13,Len(a\$)) Mid\$ (b\$,1) = NEW " MsgBox a\$ & crlf & b\$ End Sub </pre>
See Also	InStr, InStrB (on page 563) (functions), Option Compare (on page 649) (statement), Mid, Mid\$, MidB, MidB\$ (on page 605) (statements)

Mid, Mid\$, MidB, MidB\$ (statements)

Syntax	<code>Mid[\$](variable,start[,length]) = newvalue</code> <code>MidB[\$](variable,start[,length]) = newvalue</code>	
Description	Replaces one part of a string with another.	
Comments	The Mid/Mid\$ statements take the following parameters:	
	Parameter	Description
	variable	String or Variant variable to be changed.
	start	Integer specifying the character position (for <code>Mid</code> and <code>Mid\$</code>) or byte position (for <code>MidB</code> and <code>MidB\$</code>) within variable where replacement begins. If start is greater than the length of variable, then variable remains unchanged.
	length	Integer specifying the number of characters or bytes to change. If this parameter is omitted, then the entire string is changed, starting at start.
	newvalue	Expression used as the replacement. This expression must be convertible to a String .
	The resultant string is never longer than the original length of variable. With <code>Mid</code> and <code>MidB</code> , variable must be a Variant variable convertible to a String , and newvalue is any expression convertible to a string. A runtime error is generated if either variant is NULL. Statements are used to replace the following.	

	Statement	Replaces
	<code>MidB</code> and <code>MidB\$</code>	Sub-string of bytes
	<code>Mid</code> and <code>Mid\$</code>	Sub-string of characters
Example	<pre> This example displays a substring from the middle of a 'string variable using the Mid\$ function, replacing the 'first four characters with "NEW " using the Mid\$ statement. Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = "This is the Main string containing text." b\$ = Mid\$(a\$,13,Len(a\$)) Mid\$(b\$,1) = "NEW " End Sub </pre>	
See Also	Mid , Mid\$, MidB , MidB\$ (on page 603) (functions), Option Compare (on page 649) (statement)	

Minute (function)

Syntax	Minute (time)
Description	Returns the minute of the day encoded in the specified time parameter.
Comments	The value returned is as an Integer between 0 and 59 inclusive. The time parameter is any expression that converts to a Date .
Example	<p>This example takes the current time; extracts the hour, minute, and second; and displays them as the current time.</p> <pre> Sub Main() MsgBox "It is now minute " & Minute(Time) & " of the hour." End Sub </pre>
See Also	Day (on page 401) (function); Second (on page 705) (function); Month (on page 610) (function); Year (on page 797) (function); Hour (on page 549) (function); Weekday (on page 779) (function); DatePart (on page 398) (function).

MIRR (function)

Syntax	MIRR (ValueArray(),FinanceRate,ReinvestRate)	
Description	Returns a Double representing the modified internal rate of return for a series of periodic payments and receipts.	
Comments	The modified internal rate of return is the equivalent rate of return on an investment in which payments and receipts are financed at different rates. The interest cost of investment and the rate of interest received on the returns on investment are both factors in the calculations. The MIRR function requires the following parameters:	
	Parameter	Description
	ValueArray()	Array of Double numbers representing the payments and receipts. Positive values are payments (invested capital), and negative values are receipts (returns on investment). There must be at least one positive (investment) value and one negative (return) value.
	FinanceRate	Double representing the interest rate paid on invested monies (paid out).
	ReinvestRate	Double representing the rate of interest received on incomes from the investment (receipts).
	FinanceRate and ReinvestRate should be expressed as percentages. For example, 11 percent should be expressed as 0.11. To return the correct value, be sure to order your payments and receipts in the correct sequence.	
Example	<p>This example illustrates the purchase of a lemonade stand for \$800 financed with money borrowed at 10%. The returns are estimated to accelerate as the stand gains popularity. The proceeds are placed in a bank at 9 percent interest. The incomes are estimated (generated) over 12 months. This program first generates the income stream array in two For...Next loops, and then the modified internal rate of return is calculated and displayed. Notice that the annual rates are normalized to monthly rates by dividing them by 12.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10)</pre> <pre>Sub Main() Dim valu#(12) valu(1) = -800 'Initial investment</pre>	

	<pre> msg1 = valu(1) & ", " For x = 2 To 5 valu(x) = 100 + (x * 2) 'Incomes months 2-5 msg1 = msg1 & valu(x) & ", " Next x For x = 6 To 12 valu(x) = 100 + (x * 10) 'Incomes months 6-12 msg1 = msg1 & valu(x) & ", " Next x retrn# = MIRR(valu,.1/12,.09/12) 'Note: normalized annual rates msg1 = "The values: " & crlf & msg1 & crlf & crlf MsgBox msg1 & "Modified rate: " & Format(retrn#,"Percent") End Sub </pre>
See Also	Fv (on page 533) (function); IRR (on page 568) (function); Npv (on page 630) (function); Pv (on page 675) (function).

MkDir (statement)

Syntax	MkDir dir\$
Description	Creates a new directory as specified by dir\$.
Example	<p>This example creates a new directory on the default drive. If this causes an error, then the error is displayed and the program terminates. If no error is generated, the directory is removed with the Rmdir statement.</p> <pre> Sub Main() On Error Resume Next MkDir "testdir" If Err <> 0 Then MsgBox "The following error occurred: " & Error(Err) Else MsgBox "Directory 'testdir' was created and is about to be removed." Rmdir "testdir" </pre>

	<pre>End If End Sub</pre>
See Also	ChDir (on page 359) (statement); ChDrive (on page 359) (statement); CurDir, CurDir\$ (on page 387) (functions); Dir, Dir\$ (on page 427) (functions); Rmdir (on page 694) (statement).

Mod (operator)

Syntax	Expression1 Mod expression2
Description	Returns the remainder of expression1 / expression2 as a whole number.
Comments	If both expressions are integers, then the result is an integer. Otherwise, each expression is converted to a Long before performing the operation, returning a Long . A runtime error occurs if the result overflows the range of a Long . If either expression is Null , then Null is returned. Empty is treated as 0.
Example	<p>This example uses the Mod operator to determine the value of a randomly selected card where card 1 is the ace (1) of clubs and card 52 is the king (13) of spades. Since the values recur in a sequence of 13 cards within 4 suits, we can use the Mod function to determine the value of any given card number.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() cval\$ = "Ace,Two,Three,Four,Five,Six,Seven,Eight,Nine,Ten,Jack,Queen,King" Randomize card% = Random(1,52) value = card% Mod 13 If value = 0 Then value = 13 CardNum\$ = Item\$(cval,value) If card% < 53 Then suit\$ = "Spades" If card% < 40 Then suit\$ = "Hearts" If card% < 27 Then suit\$ = "Diamonds" If card% < 14 Then suit\$ = "Clubs" msg1 = "Card number " & card% & " is the " msg1 = msg1 & CardNum & " of " & suit\$</pre>

	<pre>MsgBox msg1 End Sub</pre>
See Also	/ (on page 300) (operator); \ (on page 301) (operator).

Month (function)

Syntax	Month (date)
Description	Returns the month of the date encoded in the specified date parameter.
Comments	The value returned is as an Integer between 1 and 12 inclusive. The date parameter is any expression that converts to a Date .
Example	<p>This example returns the current month in a dialog box.</p> <pre>Sub Main() mons\$ = "Jan.,Feb.,Mar.,Apr.,May,Jun.,Jul.,Aug.,Sep.,Oct.,Nov.,Dec." tdate\$ = Date\$ tmonth! = Month(DateValue(tdate\$)) MsgBox "The current month is: " & Item\$(mons\$,tmonth!) End Sub</pre>
See Also	Day (on page 401) (function) Minute (on page 606) (function); Second (on page 705) (function); Year (on page 797) (function); Hour (on page 549) (function); Weekday (on page 779) (function); DatePart (on page 398) (function).

Msg.Close (method)

Syntax	Msg.Close
Description	Closes the modeless message dialog box.
Comments	Nothing will happen if there is no open message dialog box.
Example	<pre>Sub Main() Msg.Open "Printing. Please wait...",0,True,True Sleep 3000</pre>

	<pre>Msg.Close End Sub</pre>
See Also	Msg.Open (on page 611) (method); Msg.Thermometer (on page 613) (property); Msg.Text (on page 612) (property).

Msg.Open (method)

Syn-tax	Msg.Open prompt,timeout,cancel,thermometer [,XPos,YPos]	
De-scription	Displays a message in a dialog box with an optional Cancel button and thermometer.	
Com-ments	The Msg.Open method takes the following named parameters:	
	Para-meter	Description
	prompt	String containing the text to be displayed. The text can be changed using the Msg.Text property.
	time-out	Integer specifying the number of seconds before the dialog box is automatically removed. The timeout parameter has no effect if its value is 0.
	cancel	Boolean controlling whether or not a Cancel button appears within the dialog box beneath the displayed message. If this parameter is True, then a Cancel button appears. If it is not specified or False, then no Cancel button is created. If a user chooses the Cancel button at runtime, a trappable runtime error is generated (error number 18). In this manner, a message dialog box can be displayed and processing can continue as normal, aborting only when the user cancels the process by choosing the Cancel button.
	ther-mome-ter	Boolean controlling whether the dialog box contains a thermometer. If this parameter is True, then a thermometer is created between the text and the optional Cancel button. The thermometer initially indicates 0% complete and can be changed using the Msg.Thermometer property.

	XPos, YPos	Integer coordinates specifying the location of the upper left corner of the message box, in twips (twentieths of a point). If these parameters are not specified, then the window is centered on top of the application.
		Unlike other dialog boxes, a message dialog box remains open until the user selects Cancel , the timeout has expired, or the Msg.Close method is executed (this is sometimes referred to as modeless). Only a single message window can be opened at any one time. The message window is removed automatically when a script terminates. The Cancel button, if present, can be selected using either the mouse or keyboard. However, these events will never reach the message dialog unless you periodically call DoEvents from within your script.
Example		<p>This example displays several types of message boxes.</p> <pre> Sub Main() Msg.Open "Printing. Please wait...",0,True,False Sleep 3000 Msg.Close Msg.Open "Printing. Please wait...",0,True,True For x = 1 to 100 Msg.Thermometer = x Next x Sleep 1000 Msg.Close End Sub </pre>
See Also		Msg.Close (on page 610) (method); Msg.Thermometer (on page 613) (property); Msg.Text (on page 612) (property).

Msg.Text (property)

Syntax	Msg.Text [= newtext\$]
Description	Changes the text within an open message dialog box (one that was previously opened with the Msg.Open method).
Comments	The message dialog box is not resized to accommodate the new text. A runtime error will result if a message dialog box is not currently open (using Msg.Open).


Exam- ple	<p>This example creates a modeless message box, leaving room in the message text for the record number. This box contains a Cancel button.</p> <pre> Sub Main() Msg.Open "Reading Record",0,True,False For i = 1 To 100 'Read a record here. 'Update the modeless message box. Sleep 100 Msg.Text ="Reading record " & i Next i Msg.Close End Sub </pre>
See Also	<p>Msg.Close (on page 610) (method); Msg.Open (on page 611) (method); Msg.Thermometer (on page 613) (property).</p>



Msg.Thermometer (property)





Syn- tax	Msg.Thermometer [= percentage]
De- scrip- tion	Changes the percentage filled indicated within the thermometer of a message dialog box (one that was previously opened with the Msg.Open method).
Com- ments	A runtime error will result if a message box is not currently open (using Msg.Open) or if the value of percentage is not between 0 and 100 inclusive.
Exam- ple	<p>This example create a modeless message box with a thermometer and a Cancel button. This example also shows how to process the clicking of the Cancel button.</p> <pre> Sub Main() On Error Goto ErrorTrap Msg.Open "Reading records from file...",0,True,True For i = 1 To 100 'Read a record here. 'Update the modeless message box. Msg.Thermometer =i DoEvents Sleep 50 Next i </pre>


	<pre> Msg.Close On Error Goto 0 'Turn error trap off. Exit Sub ErrorTrap: If Err = 809 Then MsgBox "Cancel was pressed!" Exit Sub 'Reset error handler. End If End Sub </pre>
See Also	Msg.Close (on page 610) (method); Msg.Open (on page 611) (method); Msg.Text (on page 612) (property).

MsgBox (function)

Syn- tax	MsgBox (msg [,type] [,title])				
De- scrip- tion	Displays a message in a dialog box with a set of predefined buttons, returning an Integer representing which button was selected.				
Com- ments	<div style="border: 1px solid orange; border-radius: 10px; padding: 10px; background-color: #fff9c4;"> <p> Important: The message box has an approximate maximum allowed number of characters.</p> </div> <p>The:</p> <ul style="list-style-type: none"> • Message box is limited to 3/5ths of the screen's horizontal resolution. • Actual message will be truncated if the message box exceeds this width. <p>It is estimated that on a 1280x800 resolution approximately 128 characters fit in the message box. The estimation is based on the fact that some letters/numbers/symbols require more than one character space (e.g. M); some less (e.g. i). Therefore the exact allowed number of characters depends on what numbers/letters/symbols are used in the message.</p>				
	The <code>MsgBox</code> function takes the following parameters:				
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>msg</td> <td>Message to be displayed—any expression convertible to a String. End-of-lines can be used to separate lines (either a carriage return, line feed, or both). If a given</td> </tr> </tbody> </table>	Parameter	Description	msg	Message to be displayed—any expression convertible to a String . End-of-lines can be used to separate lines (either a carriage return, line feed, or both). If a given
Parameter	Description				
msg	Message to be displayed—any expression convertible to a String . End-of-lines can be used to separate lines (either a carriage return, line feed, or both). If a given				

		line is too long, it will be word-wrapped. If msg contains character 0, then only the characters up to the character 0 will be displayed. The width and height of the dialog box are sized to hold the entire contents of msg. A runtime error is generated if msg is Null .		
type		Integer specifying the type of dialog box (see below).		
title		Caption of the dialog box. This parameter is any expression convertible to a String . If it is omitted, then the script is used. A runtime error is generated if title is Null .		
		The MsgBox function returns one of the following values:		
	Constant	Value	The following is clicked	
	ebOK	1	OK	
	ebCancel	2	Cancel	
	ebAbort	3	Abort	
	ebRetry	4	Retry	
	ebIgnore	5	Ignore	
	ebYes	6	Yes	
	ebNo	7	No	
	The type parameter is the sub of any of the following values:			
	Constant	Value	Displays	
	ebOKOnly	0	OK button	
	ebOKCancel	1	OK and Cancel buttons	
	ebAbortRetryIgnore	2	Abort, Retry and Ignore buttons	
	ebYesNoCancel	3	Displays Yes, No, and Cancel buttons.	
	ebYesNo	4	Yes and No buttons.	
	ebRetryCancel	5	Retry and Cancel buttons	
	ebCritical	16	Stop icon	
	ebQuestion	32	Question Mark icon	

ebExclamation	48	Exclamation Point icon	
ebInformation	64	Information icon	
ebDefaultButton1	0	First button is the default button.	
ebDefaultButton2	256	Second button is the default button.	
ebDefaultButton3	512	Third button is the default button.	
ebApplicationModal	0	Application modal; the current application is suspended until the dialog box is closed.	
<p>The default value for type is 0 (display only the OK button, making it the default). Breaking Text across Lines The msg parameter can contain end-of-line characters, forcing the text that follows to start on a new line. The following example shows how to display a string on two lines:</p> <pre>MsgBox "This is on" + Chr(13) + Chr(10) + "two lines."</pre> <p>The carriage-return or line-feed characters can be used by themselves to designate an end-of-line.</p>			
<pre>r = MsgBox("Hello, World")</pre>  <pre>r = MsgBox("Hello, World",ebYesNoCancel Or ebDefaultButton1)</pre>  <pre>r = MsgBox("Hello, World",ebYesNoCancel Or ebDefaultButton1 Or ebCritical)</pre>			

	
<p>Example</p>	<pre> Sub Main() MsgBox "This is a simple message box." MsgBox "This is a message box with a title and an icon.",_ ebExclamation,"Simple" MsgBox "This message box has OK and Cancel buttons.",_ ebOkCancel,"MsgBox" MsgBox "This message box has Abort, Retry, and Ignore buttons.",_ ebAbortRetryIgnore,"MsgBox" MsgBox "This message box has Yes, No, and Cancel buttons.",_ ebYesNoCancel Or ebDefaultButton2,"MsgBox" MsgBox "This message box has Yes and No buttons.",ebYesNo,"MsgBox" MsgBox "This message box has Retry and Cancel buttons.",_ ebRetryCancel,"MsgBox" MsgBox "This message box is system modal!",ebSystemModal End Sub </pre>
<p>See Also</p>	<p>AskBox\$ (on page 333) (function); AskPassword\$ (on page 335) (function); InputBox, InputBox\$ (on page 562) (functions); OpenFilename\$ (on page 646) (function); SaveFilename\$ (on page 699) (function); SelectBox (on page 709) (function); AnswerBox (on page 310) (function).</p>
<p>Note</p>	<p><code>MsgBox</code> displays all text in its dialog box in 8-point MS Sans Serif.</p>

MsgBox (statement)

<p>Syntax</p>	<p>MsgBox msg [,type] [,title]</p>
<p>Description</p>	<p>This command is the same as the MsgBox function, except that the statement form does not return a value. See MsgBox (function).</p>

Ex- am- ple	<pre> Sub Main() MsgBox "This is text displayed in a message box." 'Display text. MsgBox "The result is: " & (10 * 45) 'Display a number. End Sub </pre>
See Also	AskBox\$ (on page 333) (function); AskPassword\$ (on page 335) (function); Input, Input\$, InputB, InputB\$ (on page 560) (functions); OpenFilename\$ (on page 646) (function); SaveFilename\$ (on page 699) (function); SelectBox (on page 709) (function); AnswerBox (on page 310) (function).

N

N

Name (statement)
Named Parameters (topic)
Net.AddCon (method)
Net.Browse\$ (method)
Net.CancelCon (method)
Net.GetCaps (method)
Net.GetCon\$ (method)
Net.User\$ (property)
New (keyword)
Not (operator)
Nothing (constant)
Now (function)
NPer (function)
Npv (function)
Null (constant)

Name (statement)

Syntax	Name oldfile\$ As newfile\$
Description	Renames a file.
Comments	<p>Each parameter must specify a single filename. Wildcard characters such as * and ? are not allowed. Some platforms allow naming of files to different directories on the same physical disk volume. For example, the following rename will work under Windows:</p> <pre>Name "c:\samples\mydoc.txt" As "c:\backup\doc\mydoc.bak"</pre> <p>You cannot rename files across physical disk volumes. For example, the following will error under Windows:</p> <pre>Name "c:\samples\mydoc.txt" As "a:\mydoc.bak" 'This will error!</pre> <p>To rename a file to a different physical disk, you must first copy the file, then erase the original:</p> <pre>FileCopy "c:\samples\mydoc.txt","a:\mydoc.bak" 'Make a copy Kill "c:\samples\mydoc.txt" 'Delete the original</pre>
Example	<p>This example creates a file called test.dat and then renames it to test2.dat.</p> <pre>Sub Main() oldfile\$ = "test.dat" newfile\$ = "test2.dat" On Error Resume Next If FileExists(oldfile\$) Then Name oldfile\$ As newfile\$ If Err <> 0 Then msg1 = "The following error occurred: " & Error(Err) Else msg1 = "" & oldfile\$ & " was renamed to " & newfile\$ & "" End If Else Open oldfile\$ For Output As #1 Close Name oldfile\$ As newfile\$ If Err <> 0 Then</pre>

	<pre> msg1 = "" & oldfile\$ & "' not created. The following error occurred: " & Error(Err) Else msg1 = "" & oldfile\$ & "' was created and renamed to '" & newfile\$ & "'" End If End If MsgBox msg1 End Sub </pre>
See Also	Kill (on page 579) (statement), FileCopy (on page 513) (statement).

Named Parameters (topic)

Many language elements in BasicScript support named parameters. Named parameters allow you to specify parameters to a function or subroutine by name rather than in adherence to a predetermined order. The following table contains examples showing various calls to **MsgBox** both using parameter by both name and position.

By Name	MsgBox Prompt:= "Hello, world."
By Position	MsgBox "Hello, world."
By Name	MsgBox Title:="Title", Prompt:="Hello, world."
By Position	MsgBox "Hello, world" ,,"Title"
By Name	MsgBox HelpFile:="BASIC.HLP", _
	Prompt:="Hello, world.", HelpContext:=10
By Position	MsgBox "Hello, world." ,,"BASIC.HLP",10

Using named parameter makes your code easier to read, while at the same time removes you from knowing the order of parameter. With function that require many parameters, most of which are optional (such as **MsgBox**), code becomes significantly easier to write and maintain.

When supported, the names of the named parameter appear in the description of that language element.

When using named parameter, you must observe the following rules:

- Named parameter must use the parameter name as specified in the description of that language element. Unrecognized parameter names cause compiler errors.
- All parameters, whether named or positional, are separated by commas.
- The parameter name and its associated value are separated with :=
- If one parameter is named, then all subsequent parameter must also be named as shown below:

```
MsgBox "Hello, world", Title:="Title" 'OK
MsgBox Prompt:="Hello, world.",,"Title" 'WRONG!!!
```

Net.AddCon (method)

Syntax	Net.AddCon NetPath,Password,LocalName [, [UserName] [,isPermanent]]	
Description	Redirects a local device (a disk drive or printer queue) to the specified shared device or remote server. The new syntax does not affect previously compiled code. If Password is not specified, then the default password is used. If empty, then no password is used. If LocalName is not specified, then the a connection is made to the network resource without redirecting the local device. The UserName parameter specifies the name of the user making the connection. If UserName is not specified, then the default user for that process is used. The isPermanent parameter specifies whether the connection should be restored during subsequent logon operations. Only a successful connection will persist in this manner.	
Comments	The Net.AddCon method takes the following parameters:	
	Parameter	Description
	net-path\$	String containing the name of the shared device or the name of a remote server. This parameter can contain the name of a shared printer queue (such as that returned by Net.Browse[1]) or the name of a network path (such as that returned by Net.Browse[0]).
	password\$	String containing the password for the given device or server. This parameter is mainly used to specify the password on a remote server.
	local-name\$	String containing the name of the local device being redirected, such as "LPT1" or "D:".

	A runtime error will result if no network is present.
Example	<p>This example sets N: so that it refers to the network path SYS:\PUBLIC.</p> <pre>Sub Main() Net.AddCon "SYS:\PUBLIC", "", "N:" End Sub</pre>
See Also	Net.CancelCon (on page 623) (method); Net.GetCon\$ (on page 625) (method)

Net.Browse\$ (method)

Syntax	Net.Browse\$ (type)										
Description	Calls the currently installed network's browse dialog box, requesting a particular type of information.										
Comments	The type parameter is an Integer specifying the type of dialog box to display:										
	<table border="1"> <thead> <tr> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>If type is 0, then this method displays a dialog box that allows the user to browse network volumes and directories. Choosing OK returns the completed pathname as a String.</td> </tr> <tr> <td>1</td> <td>If type is 1, then this function displays a dialog box that allows the user to browse the network's printer queues. Choosing OK returns the complete name of that printer queue as a String. This string is the same format as required by the Net.AddCon method.</td> </tr> <tr> <td>2</td> <td>Display the Disconnect dialog for disk resources.</td> </tr> <tr> <td>3</td> <td>Display the Disconnect dialog for printer resources.</td> </tr> </tbody> </table>	Type	Description	0	If type is 0, then this method displays a dialog box that allows the user to browse network volumes and directories. Choosing OK returns the completed pathname as a String.	1	If type is 1, then this function displays a dialog box that allows the user to browse the network's printer queues. Choosing OK returns the complete name of that printer queue as a String. This string is the same format as required by the Net.AddCon method.	2	Display the Disconnect dialog for disk resources.	3	Display the Disconnect dialog for printer resources.
Type	Description										
0	If type is 0, then this method displays a dialog box that allows the user to browse network volumes and directories. Choosing OK returns the completed pathname as a String.										
1	If type is 1, then this function displays a dialog box that allows the user to browse the network's printer queues. Choosing OK returns the complete name of that printer queue as a String. This string is the same format as required by the Net.AddCon method.										
2	Display the Disconnect dialog for disk resources.										
3	Display the Disconnect dialog for printer resources.										
	This dialog box differs depending on the type of network installed. A runtime error will result if no network is present.										
Example	<p>This example retrieves a valid network path.</p> <pre>Sub Main() s\$ = Net.Browse\$(0) If s\$ <> "" Then MsgBox "The following network path was selected: " & s\$ End If End Sub</pre>										

```

Else
    MsgBox "Dialog box was canceled."
End If

End Sub
    
```

Net.CancelCon (method)

Syntax	Net.CancelCon Connection [,isForce] [,isPermanent]]	
Description	The isForce parameter is True if missing or omitted. The isPermanent parameter indicates if the disconnection should persist to subsequent logon operations. On all platforms, the Connection parameter specifies what is to be disconnected. If Connection specifies a local device, then only that device is disconnected. If Connection specifies a remote device, then all local devices attached to that remote device are disconnected. Cancels a network connection.	
Comments	The Net.CancelCon method takes the following parameters:	
	Parameter	Description
	connection\$	String containing the name of the device to cancel, such as "LPT1" or "D:".
	isForce	<p>Boolean specifying whether to force the cancellation of the connection if there are open files or open print jobs.</p> <ul style="list-style-type: none"> • If this parameter is True, then this method will close all open files and open print jobs before the connection is closed. • If this parameter is False, this the method will issue a runtime error if there are any open files or open print jobs.
	A runtime error will result if no network is present.	
Example	<p>This example deletes the drive mapping associated with drive N:.</p> <pre> Sub Main() Net.CancelCon "N:" End Sub </pre>	

See Also	Net.AddCon (on page 621) (method); Net.GetCon\$ (on page 625) (method).
----------	---

Net.GetCaps (method)

Syn- tax	<code>Net.GetCaps(type [,localname\$])</code>	
De- scrip- tion	Returns an Integer specifying information about the network and its capabilities.	
Com- ments	The <code>Net.GetCaps</code> method takes the following parameters:	
	Para- meter	Description
	type	Integer specifying what type of information to retrieve. This parameter is different from platform to platform.
	local- name\$	String specifying the name of the local device to which is attached to the network device to be queried. If this parameter is missing, then information about the first network device is returned.
	A runtime error will result if no network is present.	
	The type parameter can be any of the following values:	
	Val- ue	Description
	1	Always returns 0.
	2	Network type:
	0	No network is installed.
	1	Microsoft Network
	2	Microsoft LAN Manager.
	3	Novell NetWare.
	4	Banyan Vines

	5	10Net
	6	Locus
	7	SunSoft PC NFS.
	8	LanStep
	9	9 Titles.
	10	Articom Lantastic
	11	IBM AS/400
	12	FTP Software FTP NFS.
	13	DEC Pathworks
Version of the network with the major version in the high byte and the minor version in the low byte: Major = Net.GetCaps(2) \ 256 Minor = Net.GetCaps(2) And &H00FF		
Example	<pre> Sub Main() 'This example checks the type of network. If Net.GetCaps(2) = 768 Then _ MsgBox "This is a Novell network." 'This checks whether the net supports retrieval of the 'user name. If Net.GetCaps(4) And 1 Then _ MsgBox "User name is: " & Net.User\$ 'This checks whether this net supports the Browse dialog 'boxes. If Net.GetCaps(6) And &H0010 Then MsgBox Net.Browse\$(1) End Sub </pre>	

Net.GetCon\$ (method)

Syntax	Net.GetCon\$ (localname\$)
--------	-----------------------------------

De- scrip- tion	Returns the name of the network resource associated with the specified redirected local device.
Com- ments	The localname\$ parameter specifies the name of the local device, such as "LPT1" or "D:". The function returns a zero-length string if the specified local device is not redirected. A runtime error will result if no network is present.
Exam- ple	This example finds out where drive Z is mapped. <pre>Sub Main() NetPath\$ = Net.GetCon\$("Z:") MsgBox "Drive Z is mapped as " & NetPath\$ End Sub</pre>
See Also	Net.CancelCon (on page 623) (method); Net.AddCon (on page 621) (method).

Net.User\$ (property)

Syn- tax	Net.User\$ [[LocalName]]
De- scrip- tion	Returns the name of the user on the network.
Com- ments	A runtime error is generated if the network is not installed. The LocalName parameter is the name of the local device that the user has made a connection to. If this parameter is omitted, then the name of the current user of the process is used. If Localname is a network name and the user is connected to that resource using different names, the network provider may not be able to resolve which user name to return. In this case, the provider may make an arbitrary choice from the possible user names.
Exam- ple	<pre>Sub Main() 'This example tells the user who he or she is. MsgBox "You are " & Net.User\$ 'This example makes sure this capability is supported. If Net.GetCaps(4) And 1 Then MsgBox "You are " & _ Net.User\$ End Sub</pre>

New (keyword)

Syn-tax 1	Dim ObjectVariable As New ObjectType
Syn-tax 2	Set ObjectVariable = New ObjectType
Description	Creates a new instance of the specified object type, assigning it to the specified object variable.
Comments	The New keyword is used to declare a new instance of the specified data object. This keyword can only be used with data object types. At runtime, the application or extension that defines that object type is notified that a new object is being defined. The application responds by creating a new physical object (within the appropriate context) and returning a reference to that object, which is immediately assigned to the variable being declared. When that variable goes out of scope (that is, the Sub or Function procedure in which the variable is declared ends), the application is notified. The application then performs some appropriate action, such as destroying the physical object.
See Also	Dim (on page 426) (statement); Set (on page 714) (statement).

Not (operator)

Syn-tax	Not expression				
Description	Returns either a logical or binary negation of expression.				
Comments	The result is determined as shown in the following table:				
	<table border="1"> <thead> <tr> <th>If the expression is</th> <th>Then the result is</th> </tr> </thead> <tbody> <tr> <td>TRUE</td> <td>FALSE</td> </tr> </tbody> </table>	If the expression is	Then the result is	TRUE	FALSE
If the expression is	Then the result is				
TRUE	FALSE				

	FALSE	TRUE
	NULL	NULL
	Any numeric type	A binary negation of the number. If the number is an Integer , then an Integer is returned. Otherwise, the expression is first converted to a Long , then a binary negation is performed, returning a Long .
	empty	Treated as a Long value 0.
Example	<p>This example demonstrates the use of the Not operator in comparing logical expressions and for switching a True/False toggle variable.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a = False b = True If (Not a and b) Then msg1 = "a = False, b = True" & crlf toggle% = True msg1 = msg1 & "toggle% is now " & CBool(toggle%) & crlf toggle% = Not toggle% msg1 = msg1 & "toggle% is now " & CBool(toggle%) & crlf toggle% = Not toggle% msg1 = msg1 & "toggle% is now " & CBool(toggle%) MsgBox msg1 End Sub </pre>	
See Also	Boolean (on page 351) (data type); Comparison Operators (on page 376) (topic).	

Nothing (constant)

Description	A value indicating that an object variable no longer references a valid object.
Example	<pre> Sub Main() Dim a As Object If a Is Nothing Then MsgBox "The object variable references no object." Else </pre>

	<pre>MsgBox "The object variable references: " & a.Value End If End Sub</pre>
See Also	Set (on page 714) (statement); Object (on page 633) (data type).

Now (function)

Syntax	Now[()]
Description	Returns a Date variant representing the current date and time.
Example	<p>This example shows how the Now function can be used as an elapsed-time counter.</p> <pre>Sub Main() t1# = Now MsgBox "Wait a while and click OK." t2# = Now t3# = Second(t2#) - Second(t1#) MsgBox "Elapsed time was: " & t3# & " seconds." End Sub</pre>
See Also	Date, Date\$ (on page 393) (functions); Time, Time\$ (on page 755) (functions).

NPer (function)

Syntax	NPer (Rate,Pmt,Pv,Fv,Due)		
Description	Returns the number of periods for an annuity based on periodic fixed payments and a constant rate of interest.		
Comments	An annuity is a series of fixed payments paid to or received from an investment over a period of time. Examples of annuities are mortgages, retirement plans, monthly savings plans, and term loans. The NPer function requires the following parameters:		
	<table border="1"> <tr> <td>Parameter</td> <td>Description</td> </tr> </table>	Parameter	Description
Parameter	Description		

	Rate	Double representing the interest rate per period. If the periods are monthly, be sure to normalize annual rates by dividing them by 12.
	Pmt	Double representing the amount of each payment or income. Income is represented by positive values, whereas payments are represented by negative values.
	Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan, and the future value (see below) would be zero.
	Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be zero, and the present value would be the amount of the loan.
	Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 indicates payment at the start of each period.
Positive numbers represent cash received, whereas negative numbers represent cash paid out.		
Example	<p>This example calculates the number of \$100.00 monthly payments necessary to accumulate \$10,000.00 at an annual rate of 10%. Payments are made at the beginning of the month.</p> <pre> Sub Main() ag# = NPer((.10/12),100,0,10000,1) MsgBox "The number of monthly periods is: " & Format(ag#,"Standard") End Sub </pre>	
See Also	IPmt (on page 566) (function); Pmt (on page 661) (function); PPmt (on page 663) (function); Rate (on page 680) (function).	

Npv (function)

Syntax	Npv (Rate,ValueArray ())	
Description	Returns the net present value of an annuity based on periodic payments and receipts, and a discount rate.	
Comments	The Npv function requires the following parameters:	
	Para-	Description

	me- ter	
	Rate	Double that represents the interest rate over the length of the period. If the values are monthly, annual rates must be divided by 12 to normalize them to monthly rates.
	Val- ue- Ar- ray()	Array of Double numbers representing the payments and receipts. Positive values are payments, and negative values are receipts. There must be at least one positive and one negative value.
		Positive numbers represent cash received, whereas negative numbers represent cash paid out. For accurate results, be sure to enter your payments and receipts in the correct order because Npv uses the order of the array values to interpret the order of the payments and receipts. If your first cash flow occurs at the beginning of the first period, that value must be added to the return value of the Npv function. It should not be included in the array of cash flows. Npv differs from the Pv function in that the payments are due at the end of the period and the cash flows are variable. Pv 's cash flows are constant, and payment may be made at either the beginning or end of the period.
Exam- ple		<p>This example illustrates the purchase of a lemonade stand for \$800 financed with money borrowed at 10%. The returns are estimated to accelerate as the stand gains popularity. The incomes are estimated (generated) over 12 months. This program first generates the income stream array in two For...Next loops, and then the net present value (Npv) is calculated and displayed. Note normalization of the annual 10% rate.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim valu#(12) valu(1) = -800 'Initial investment msg1 = valu(1) & ", " For x = 2 To 5 'Months 2-5 valu(x) = 100 + (x * 2) msg1 = msg1 l& valu(x) & ", " Next x For x = 6 To 12 'Months 6-12 valu(x) = 100 + (x * 10) 'Accelerated income msg1 = msg1 & valu(x) & ", " Next x NetVal# = NPV(.10/12),valu) </pre>

	<pre>msg1 = "The values:" & crlf & msg1 & crlf & crlf MsgBox msg1 & "Net present value: " & Format(NetVal#, "Currency") End Sub</pre>
See	Fv (on page 533) (function); IRR (on page 568) (function); MIRR (on page 606) (function);
Also	Pv (on page 675) (function).

Null (constant)

De- scrip- tion	Represents a variant of VarType 1 .
Com- ments	The Null value has special meaning indicating that a variable contains no data. Most numeric operators return Null when either of the arguments is Null . This "propagation" of Null makes it especially useful for returning error values through a complex expression. For example, you can write functions that return Null when an error occurs, then call this function within an expression. You can then use the IsNull function to test the final result to see whether an error occurred during calculation. Since variants are Empty by default, the only way for Null to appear within a variant is for you to explicitly place it there. Only a few functions return this value.
Exam- ple	<pre>Sub Main() Dim a As Variant a = Null If IsNull(a) Then MsgBox "The variable is Null." MsgBox "The VarType of a is: " & VarType(a) 'Should display 1. End Sub</pre>

0

0

Object (data type)
Objects (topic)
Oct, Oct\$ (functions)
OKButton (statement)
On Error (statement)

Open (statement)
OpenFilename\$ (function)
Operator Precedence (topic)
Operator Precision(topic)
Option Base (statement)
Option Compare (statement)
Option CStrings (statement)
Option Default (statement)
Option Explicit (statement)
OptionButton (statement)
OptionGroup (statement)
Or (operator)

Object (data type)

Syn- tax	Object
De- scrip- tion	A data type used to declare OLE automation variables.
Com- ments	The Object type is used to declare variables that reference objects within an application using OLE automation. Each object is a 4-byte (32-bit) value that references the object internally. The value 0 (or Nothing) indicates that the variable does not reference a valid object, as is the case when the object has not yet been given a value. Accessing properties or methods of such Object variables generates a runtime error.
	<p>Using Objects Object variables are declared using the Dim, Public, or Private statement:</p> <pre>Dim MyApp As Object</pre> <p>Object variables can be assigned values (thereby referencing a real physical object) using the Set statement:</p>

	<pre>Set MyApp = CreateObject("phantom.application") Set MyApp = Nothing</pre> <p>Properties of an Object are accessed using the dot (.) separator:</p> <pre>MyApp.Color = 10 i% = MyApp.Color</pre> <p>Methods of an Object are also accessed using the dot (.) separator:</p> <pre>MyApp.Open "sample.txt" isSuccess = MyApp.Save("new.txt",15)</pre>
	<p>Automatic Destruction The Basic Control Engine keeps track of the number of variables that reference a given object so that the object can be destroyed when there are no longer any references to it:</p> <pre>Sub Main() 'Number of references to object Dim a As Object '0 Dim b As Object '0 Set a = CreateObject("phantom.application") '1 Set b = a '2 Set a = Nothing '1 End Sub '0 (object destroyed)</pre> <p>Note An OLE automation object is instructed by the Basic Control Engine to destroy itself when no variables reference that object. However, it is the responsibility of the OLE automation server to destroy it. Some servers do not destroy their objects—usually when the objects have a visual component and can be destroyed manually by the user.</p>
See Also	<p>Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Integer (on page 566) (data type); Long (on page 598) (data type); Single (on page 718) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement).</p>

Objects (topic)

The Basic Control Engine defines two types of objects: data objects and OLE automation objects.

Syntactically, these are referenced in the same way.

What Is an Object

An object in the Basic Control Engine is an encapsulation of data and routines into a single unit. The use of objects in the Basic Control Engine has the effect of grouping together a set of functions and data items that apply only to a specific object type.

Objects expose data items for programmability called properties. For example, a **sheet** object may expose an integer called **NumColumns**. Usually, properties can be both retrieved (get) and modified (set).

Objects also expose internal routines for programmability called methods. In the Basic Control Engine, an object method can take the form of a function or a subroutine. For example, a OLE automation object called **MyApp** may contain a method subroutine called **Open** that takes a single argument (a filename), as shown below:

```
MyApp.Open "c:\files\sample.txt"
```

Declare Object Variables

In order to gain access to an object, you must first declare an object variable using either **Dim**, **Public**, or **Private**:

```
Dim o As Object 'OLE automation object
```

Initially, objects are given the value **0** (or **Nothing**). Before an object can be accessed, it must be associated with a physical object.

Assign a Value to an Object Variable

An object variable must reference a real physical object before accessing any properties or methods of that object. To instantiate an object, use the **Set** statement.

```
Dim MyApp As Object
Set MyApp = CreateObject("Server.Application")
```

Access Object Properties

Once an object variable has been declared and associated with a physical object, it can be modified using the Basic Control Engine code. Properties are syntactically accessible using the dot operator, which separates an object name from the property being accessed:

```
MyApp.BackgroundColor = 10
i% = MyApp.DocumentCount
```

Properties are set using the Basic Control Engine normal assignment statement:

```
MyApp.BackgroundColor = 10
```

Object properties can be retrieved and used within expressions:

```
i% = MyApp.DocumentCount + 10
MsgBox "Number of documents = " & MyApp.DocumentCount
```

Access Object Methods

Like properties, methods are accessed via the dot operator. Object methods that do not return values behave like subroutines in the Basic Control Engine (that is, the arguments are not enclosed within parentheses):

```
MyApp.Open "c:\files\sample.txt",True,15
```

Object methods that return a value behave like function calls in the Basic Control Engine. Any arguments must be enclosed in parentheses:

```
If MyApp.DocumentCount = 0 Then MsgBox "No open documents."
NumDocs = app.count(4,5)
```

There is no syntactic difference between calling a method function and retrieving a property value, as shown below:

```
variable = object.property(arg1,arg2)  variable = object.method(arg1,arg2)
```

Compare Object Variables

The values used to represent objects are meaningless to the script in which they are used, with the following exceptions:

- Objects can be compared to each other to determine whether they refer to the same object.
- Objects can be compared with **Nothing** to determine whether the object variable refers to a valid object.

Object comparisons are accomplished using the **Is** operator:

```
If a Is b Then MsgBox "a and b are the same object."
If a Is Nothing Then MsgBox "a is not initialized."
If b Is Not Nothing Then MsgBox "b is in use."
```

Collections

A collection is a set of related object variables. Each element in the set is called a member and is accessed via an index, either numeric or text, as shown below:

```
MyApp.Toolbar.Buttons(0)

MyApp.Toolbar.Buttons("Tuesday")
```

It is typical for collection indexes to begin with **0**.

Each element of a collection is itself an object, as shown in the following examples:

```
Dim MyToolbarButton As Object

Set MyToolbarButton = MyApp.Toolbar.Buttons("Save")

MyApp.Toolbar.Buttons(1).Caption = "Open"
```

The collection itself contains properties that provide you with information about the collection and methods that allow navigation within that collection:

```
Dim MyToolbarButton As Object

NumButtons% = MyApp.Toolbar.Buttons.Count

MyApp.Toolbar.Buttons.MoveNext

MyApp.Toolbar.Buttons.FindNext "Save"

For i = 1 To MyApp.Toolbar.Buttons.Count

    Set MyToolbarButton = MyApp.Toolbar.Buttons(i)

    MyToolbarButton.Caption = "Copy"

Next i
```

Predefined Objects

The Basic Control Engine predefines a few objects for use in all scripts. These are:

```
Clipboard System HWND

Net Basic Screen
```

Oct, Oct\$ (functions)

Syn- tax	Oct[\$] (number)
-------------	-------------------------

De- scrip- tion	Returns a String containing the octal equivalent of the specified number.
Com- ments	Oct\$ returns a String , whereas Oct returns a String variant. The returned string contains only the number of octal digits necessary to represent the number. The number parameter is any numeric expression. If this parameter is Null , then Null is returned. Empty is treated as 0. The number parameter is rounded to the nearest whole number before converting to the octal equivalent.
Exam- ple	<p>This example accepts a number and displays the decimal and octal 'equivalent until the input number is 0 or invalid.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Do xs\$ = InputBox("Enter a number to convert:", "Octal Convert") x = Val(xs\$) If x <> 0 Then MsgBox "Decimal: " & x & " Octal: " & Oct(x) Else MsgBox "Goodbye." End If Loop While x <> 0 End Sub </pre>
See Also	Hex , Hex\$ (on page 547) (functions).

OKButton (statement)

Syn- tax	OKButton X,Y,width,height [,.Identifier]
De- scrip- tion	Creates an OK button within a dialog box template.
Com- ments	This statement can only appear within a dialog box template (that is, between the Begin Dialog and End Dialog statements). The OKButton statement accepts the following parameters:

	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).
	If the DefaultButton parameter is not specified in the Dialog statement, the OK button will be used as the default button. In this case, the OK button can be selected by pressing Enter on a nonbutton control. A dialog box template must contain at least one OKButton , CancelButton , or PushButton statement (otherwise, the dialog box cannot be dismissed).	
Example	<p>This example shows how to use the OK and Cancel buttons within a dialog box template and how to detect which one closed the dialog box.</p> <pre data-bbox="305 940 1427 1663"> Sub Main() Begin Dialog QuitDialogTemplate 16,32,116,64,"Quit" Text 4,8,108,8,"Are you sure you want to exit?" CheckBox 32,24,63,8,"Save Changes",.SaveChanges OKButton 12,40,40,14 CancelButton 60,40,40,14 End Dialog Dim QuitDialog As QuitDialogTemplate rc% = Dialog(QuitDialog) Select Case rc% Case -1 MsgBox "OK was pressed!" Case 1 MsgBox "Cancel was pressed!" End Select End Sub </pre>	
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OptionButton (on page 652) (statement); OptionGroup (on page	

[653](#)) (statement); [Picture \(on page 657\)](#) (statement); [PushButton \(on page 671\)](#) (statement); [Text \(on page 752\)](#) (statement); [TextBox \(on page 753\)](#) (statement); [Begin \(on page 348\)](#) [Dialog \(on page 348\)](#) (statement), [PictureBox \(on page 659\)](#) (statement).

On Error (statement)

Syntax	On Error {Goto label Resume Next Goto 0}
Description	Defines the action taken when a trappable runtime error occurs.
Comments	The form On Error Goto label causes execution to transfer to the specified label when a runtime error occurs. The form On Error Resume Next causes execution to continue on the line following the line that caused the error. The form On Error Goto 0 causes any existing error trap to be removed. If an error trap is in effect when the script ends, then an error will be generated. An error trap is only active within the subroutine or function in which it appears. Once an error trap has gained control, appropriate action should be taken, and then control should be resumed using the Resume statement. The Resume statement resets the error handler and continues execution. If a procedure ends while an error is pending, then an error will be generated. (The Exit Sub or Exit Function statement also resets the error handler, allowing a procedure to end without displaying an error message.)
	<p>Errors within an Error Handler If an error occurs within the error handler, then the error handler of the caller (or any procedure in the call stack) will be invoked. If there is no such error handler, then the error is fatal, causing the script to stop executing. The following statements reset the error state (that is, these statements turn off the fact that an error occurred):</p> <pre>Resume Err=-1</pre> <p>The Resume statement forces execution to continue either on the same line or on the line following the line that generated the error. The Err=-1 statement allows explicit resetting of the error state so that the script can continue normal execution without resuming at the statement that caused the error condition. The On Error statement will not reset the error. Thus, if an On Error statement occurs within an error handler, it has the effect of changing the location of a new error handler for any new errors that may occur once the error has been reset.</p>
Example	This example will demonstrate three types of error handling. The first case simply by-passes an expected error and continues with program operation. The second case creates an error branch

that jumps to a common error handling routine that processes incoming errors, clears the error (with the Resume statement) and resumes program execution. The third case clears all internal error handling so that execution will stop when the next error is encountered.

```

Sub Main()

    Dim x%

    a = 10000

    b = 10000

    On Error Goto Pass 'Branch to this label on error.

    Do

        x% = a * b

    Loop

Pass:

    Err = -1 'Clear error status.

    MsgBox "Cleared error status and continued."

    On Error Goto Overflow 'Branch to new error routine on any

    x% = 1000 'subsequent errors.

    x% = a * b

    x% = a / 0

    On Error Resume Next 'Pass by any following errors until

    x% = 1000 'another On Error statement is

    x% = a * b 'encountered.

    On Error Goto 0 'Clear error branching.

    x% = a * b 'Program will stop here.

    Exit Sub 'Exit before common error routine.

Overflow: 'Beginning of common error routine.

    If Err = 6 then

        MsgBox "Overflow Branch."

    Else

        MsgBox Error(Err)

    End If

    Resume Next

End Sub

```

See	Error Handling (on page 495) (topic); Error (on page 494) (statement); Resume (on page 691) (statement).
Also	

Open (statement)

Syn- tax	Open <i>filename</i> \$ [For mode] [Access accessmode] [<i>lock</i>] As [#] <i>filenumber</i> _ [<i>Len = reclen</i>]											
De- scrip- tion	Opens a file for a given mode, assigning the open file to the supplied filenumber.											
Com- ments	<p>The <i>filename</i>\$ parameter is a string expression that contains a valid filename.</p> <p>The filenumber parameter is a number between 1 and 255. The FreeFile function can be used to determine an available file number.</p> <p>The mode parameter determines the type of operations that can be performed on that file:</p> <table border="1"> <thead> <tr> <th>File Mode</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Input</td> <td>Opens an existing file for sequential input (<i>filename</i>\$ must exist). The value of <i>accessmode</i>, if specified, must be Read.</td> </tr> <tr> <td>Output</td> <td>Opens an existing file for sequential output, truncating its length to zero, or creates a new file. The value of <i>accessmode</i>, if specified, must be Write.</td> </tr> <tr> <td>Append</td> <td>Opens an existing file for sequential output, positioning the file pointer at the end of the file, or creates a new file. The value of <i>accessmode</i>, if specified, must be Read Write.</td> </tr> <tr> <td>Random</td> <td>Opens an existing file for record I/O or creates a new file. Existing random files are truncated only if <i>accessmode</i> is Write. The <i>reclen</i> parameter determines the record length for I/O operations.</td> </tr> </tbody> </table>		File Mode	Description	Input	Opens an existing file for sequential input (<i>filename</i> \$ must exist). The value of <i>accessmode</i> , if specified, must be Read .	Output	Opens an existing file for sequential output, truncating its length to zero, or creates a new file. The value of <i>accessmode</i> , if specified, must be Write .	Append	Opens an existing file for sequential output, positioning the file pointer at the end of the file, or creates a new file. The value of <i>accessmode</i> , if specified, must be Read Write .	Random	Opens an existing file for record I/O or creates a new file. Existing random files are truncated only if <i>accessmode</i> is Write . The <i>reclen</i> parameter determines the record length for I/O operations.
File Mode	Description											
Input	Opens an existing file for sequential input (<i>filename</i> \$ must exist). The value of <i>accessmode</i> , if specified, must be Read .											
Output	Opens an existing file for sequential output, truncating its length to zero, or creates a new file. The value of <i>accessmode</i> , if specified, must be Write .											
Append	Opens an existing file for sequential output, positioning the file pointer at the end of the file, or creates a new file. The value of <i>accessmode</i> , if specified, must be Read Write .											
Random	Opens an existing file for record I/O or creates a new file. Existing random files are truncated only if <i>accessmode</i> is Write . The <i>reclen</i> parameter determines the record length for I/O operations.											

If the *mode* parameter is missing, then **Random** is used.

The *accessmode* parameter determines what type of I/O operations can be performed on the file:

Access	Description
Read	Opens the file for reading only. This value is valid only for files opened in Binary , Random , or Input mode.
Write	Opens the file for writing only. This value is valid only for files opened in Binary , Random , or Output mode.
Read Write	Opens the file for both reading and writing. This value is valid only for files opened in Binary , Random , or Append mode.

If the *accessmode* parameter is not specified, the following defaults are used:

File Mode	Default value for <i>access mode</i>
Input	Read
Output	Write
Append	Read Write
Binary	When the file is initially opened, access is attempted three times in the following order: <ul style="list-style-type: none"> • Read Write • Write • Read
Random	Same as Binary files.

The *lock* parameter determines what access rights are granted to other processes that attempt to open the same file. The following table describes the values for *lock*:

Lock value	Description
Shared	Another process can both read this file and write to it. (Deny none.)
Lock Read	Another process can write to this file but not read it. (Deny read.)
Lock Write	Another process can read this file but not write to it. (Deny write.)
Lock Read Write	Another process is prevented both from reading this file and from writing to it. (Exclusive.)

If *lock* is not specified, then the file is opened in **Shared** mode.

If the file does not exist and the lock parameter is specified, the file is opened twice³once to create the file and again to establish the correct sharing mode.

Files opened in **Random** mode are divided up into a sequence of records, each of the length specified by the *reclen* parameter. If this parameter is missing, then 128 is used. For files opened for sequential I/O, the *reclen* parameter specifies the size of the internal buffer used by the Basic Control Engine when performing I/O. Larger buffers mean faster file access. For **Binary** files, the *reclen* parameter is ignored.

Example

This example opens several files in various configurations.

```
Sub Main()
  Open "test.dat" For Output Access Write Lock Write As #2
  Close
  Open "test.dat" For Input Access Read Shared As #1
  Close
  Open "test.dat" For Append Access Write Lock Read Write As #3
  Close
  Open "test.dat" For Binary Access Read Write Shared As #4
  Close
  Open "test.dat" For Random Access Read Write Lock Read As #5
  Close
  Open "test.dat" For Input Access Read Shared As #6
  Close
  Kill "test.dat"
```

	<code>End Sub</code>
See	Close (on page 373) (statement); Reset (on page 690) (statement); FreeFile (on page 532) (function).
Also	

Option Default (statement)

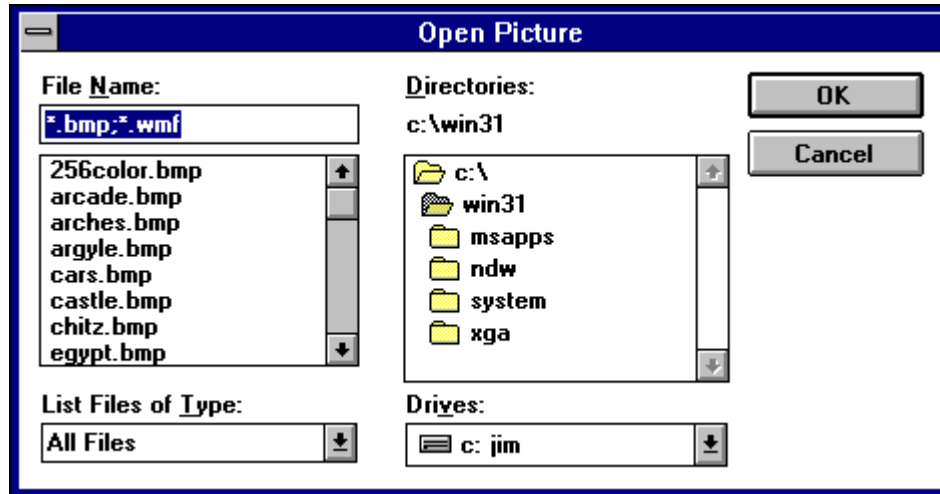
Syn- tax	<code>Option Default type</code>
De- scrip- tion	Sets the default data type of variables and function return values when not otherwise specified.
Com- ments	By default, the type of implicitly defined variables and function return values is Variant . This statement is used for backward compatibility with earlier versions of BasicScript where the default data type was Integer . This statement must appear outside the scope of all functions and subroutines. Currently, type can only be set to Integer .
Exam- ple	<pre> ' This script sets the default data type to Integer. This fact ' is used to declare the function AddIntegers which returns an ' Integer data type. Option Default Integer Function AddIntegers(a As Integer,b As Integer) Foo = a + b End Function Sub Main Dim a,b,result a = InputBox("Enter an integer:") b = InputBox("Enter an integer:") result = AddIntegers(a,b) End Sub </pre>
See	DefType (on page 421) (statement)
Also	

Option Explicit (statement)

Syn- tax	<code>Option Explicit</code>
De- scrip- tion	Prevents implicit declaration of variables and externally called procedures.
Com- ments	By default, BasicScript implicitly declares variables that are used but have not been explicitly declared with <code>Dim</code> , <code>Public</code> , or <code>Private</code> . To avoid typing errors, you may want to use <code>Option Explicit</code> to prevent this behavior. The <code>Option Explicit</code> statement also enforces explicit declaration of all externally called procedures. Once specified, all externally called procedures must be explicitly declared with the <code>Declare</code> statement.
See Also	Const (on page 378) (statement), Dim (on page 426) (statement), Public (on page 670) (statement), Private (on page 668) (statement), ReDim (on page 688) (statement), Declare (on page 412) (statement)

OpenFilename\$ (function)

Syn- tax	OpenFilename\$ [(title\$ [,extensions\$])]	
De- scrip- tion	Displays a dialog box that prompts the user to select from a list of files, returning the full path-name of the file the user selects or a zero-length string if the user selects Cancel.	
Com- ments	This function displays the standard file open dialog box, which allows the user to select a file. It takes the following parameters:	
	Parameter	Description
	Title\$	String specifying the title that appears in the dialog box's title bar. If this parameter is omitted, then "Open" is used.
	Extension\$	String specifying the available file types. If this parameter is omitted, then all files are displayed.
	<pre>e\$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF" f\$ = OpenFilename\$("Open Picture",e\$)</pre>	



Example This example asks the user for the name of a file, then proceeds to read the first line from that file.

```

Sub Main
    Dim f As String,s As String
    f$ = OpenFileDialog("Open Picture","Text Files:*.TXT")
    If f$ <> "" Then
        Open f$ For Input As #1
        Line Input #1,s$
        Close #1
        MsgBox "First line from " & f$ & " is " & s$
    End If
End Sub
    
```

See Also [MsgBox \(on page 617\)](#) (statement); [AskBox\\$ \(on page 333\)](#) (function); [AskPassword\\$ \(on page 335\)](#) (function); [InputBox, InputBox\\$ \(on page 562\)](#) (functions); [SaveFilename\\$ \(on page 699\)](#) (function); [SelectBox \(on page 709\)](#) (function); [AnswerBox \(on page 310\)](#) (function).

Notes The extensions\$ parameter must be in the following format:

```
type:ext[,ext][;type:ext[,ext]]...
```

Placeholder	Description
type	Specifies the name of the grouping of files, such as All Files .
ext	Specifies a valid file extension, such as *.BAT or *.?F? .

For example, the following are valid extensions\$ specifications:

```
"All Files:*.*"
"Documents:*.TXT,*.DOC"
"All Files:*.*;Documents:*.TXT,*.DOC"
```

Operator Precedence (topic)

The following table shows the precedence of the operators supported by the Basic Control Engine. Operations involving operators of higher precedence occur before operations involving operators of lower precedence. When operators of equal precedence occur together, they are evaluated from left to right.

Operator	Description	Precedence Order
()	Parentheses	Highest
^	Exponentiation	
-	Unary minus	
/, *	Division and multiplication	
\	Integer division	
Mod	Modulo	
+, -	Addition and subtraction	
&	String concatenation	
=, <>, >, <, <=, >=	Relational	
Like, Is	String and object comparison	
Not	Logical negation	
And	Logical or binary conjunction	
Or	Logical or binary disjunction	
Xor, Eqv, Imp	Logical or binary operators	Lowest

The precedence order can be controlled using parentheses, as shown below:

```
a = 4 + 3 * 2      'a becomes 10.
a = (4 + 3) * 2    'a becomes 14.
```

Operator Precision (topic)

When numeric, binary, logical or comparison operators are used, the data type of the result is generally the same as the data type of the more precise operand. For example, adding an **Integer** and a **Long** first converts the **Integer** operand to a **Long**, then performs a long addition, overflowing only if the result cannot be contained with a **Long**. The order of precision is shown in the following table: **Empty** Least precise **BooleanIntegerLongSingleDateDoubleCurrency** Most precise

There are exceptions noted in the descriptions of each operator. The rules for operand conversion are further complicated when an operator is used with variant data. In many cases, an overflow causes automatic promotion of the result to the next highest precise data type. For example, adding two **Integer** variants results in an **Integer** variant unless it overflows, in which case the result is automatically promoted to a **Long** variant.

Option Base (statement)

Syntax	Option Base {0 1}
Description	Sets the lower bound for array declarations.
Comments	By default, the lower bound used for all array declarations is 0. This statement must appear outside of any functions or subroutines.
Example	<pre>Option Base 1 Sub Main() Dim a(10) 'Contains 10 elements (not 11). a(1) = "Hello" MsgBox "The first element of the array is: " & a(1) End Sub</pre>
See Also	Dim (on page 426) (statement); Public (on page 670) (statement); Private (on page 668) (statement).

Option Compare (statement)

Syntax	Option Compare [Binary Text]
--------	---------------------------------------

De- scrip- tion	Controls how strings are compared.
Com- ments	<p>When Option Compare is set to Binary, then string comparisons are case-sensitive (for example, "A" does not equal "a"). When it is set to Text, string comparisons are case-insensitive (for example, "A" is equal to "a"). The default value for Option Compare is Binary. The Option Compare statement affects all string comparisons in any statements that follow the Option Compare statement. Additionally, the setting affects the default behavior of Instr, StrComp, and the Like operator. The following table shows the types of string comparisons affected by this setting: > < <> <= >= Instr StrComp Like The Option Compare statement must appear outside the scope of all subroutines and functions. In other words, it cannot appear within a Sub or Function block.</p>
Exam- ple	<p>This example shows the use of Option Compare.</p> <pre> Option Compare Binary Sub CompareBinary a\$ = "This String Contains UPPERCASE." b\$ = "this string contains uppercase." If a\$ = b\$ Then MsgBox "The two strings were compared case-insensitive." Else MsgBox "The two strings were compared case-sensitive." End If End Sub Option Compare Text Sub CompareText a\$ = "This String Contains UPPERCASE." b\$ = "this string contains uppercase." If a\$ = b\$ Then MsgBox "The two strings were compared case-insensitive." Else MsgBox "The two strings were compared case-sensitive." End If End Sub Sub Main() CompareBinary 'Calls subroutine above. </pre>

	<pre>CompareText 'Calls subroutine above. End Sub</pre>
See Also	Like (on page 587) (operator); InStr (on page 563) (function); StrComp (on page 738) (function); Comparison Operators (on page 376) (topic).

Option CStrings (statement)

Syntax	Option CStrings {On Off}		
Description	Turns on or off the ability to use C-style escape sequences within strings.		
Comments	When Option CStrings On is in effect, the compiler treats the backslash character as an escape character when it appears within strings. An escape character is simply a special character that cannot otherwise be ordinarily typed by the computer keyboard.		
	Escape	Description	Equivalent Expression
	\r	Carriage return	<code>Chr\$(13)</code>
	\n	Line feed	<code>Chr\$(10)</code>
	\a	Bell	<code>Chr\$(7)</code>
	\b	Backspace	<code>Chr\$(8)</code>
	\f	Form feed	<code>Chr\$(12)</code>
	\t	Tab	<code>Chr\$(9)</code>
	\v	Vertical tab	<code>Chr\$(11)</code>
	\0	Null	<code>Chr\$(0)</code>
	\"	Double quotation mark	<code>" "</code> or <code>Chr\$(34)</code>
	\\	Backslash	<code>Chr\$(92)</code>
	\?	Question mark	<code>?</code>
	\'	Single quotation mark	<code>'</code>
	\x hh	Hexadecimal number	<code>Chr\$(Val("&Hhh"))</code>
	\ooo	Octal number	<code>Chr\$(Val("&Oooo"))</code>

	<code>\ anycharacter</code>	Any character	<code>anycharacter</code>
	<p>With hexadecimal values, the Basic Control Engine stops scanning for digits when it encounters a nonhexadecimal digit or two digits, whichever comes first. Similarly, with octal values, the Basic Control Engine stops scanning when it encounters a nonoctal digit or three digits, whichever comes first. When Option CStrings Off is in effect, then the backslash character has no special meaning. This is the default.</p>		
Example	<pre>Option CStrings On Sub Main() MsgBox "They said, \"Watch out for that clump of grass!\"" MsgBox "First line.\r\nSecond line." MsgBox "Char A: \x41 \r\n Char B: \x42" End Sub</pre>		

OptionButton (statement)

Syntax	OptionButton X,Y,width,height,title\$ [,.Identifier]	
Description	Defines an option button within a dialog box template.	
Comments	This statement can only appear within a dialog box template (that is, between the Begin Dialog and End Dialog statements). The OptionButton statement accepts the following parameters:	
	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	title\$	String containing text that appears within the option button. This text may contain an ampersand character to denote an accelerator letter, such as "&Portrait" for Portrait , which can be selected by pressing the P accelerator.
	.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).

Example	<p>This example creates a group of option buttons.</p> <pre> Sub Main() Begin Dialog PowerTemplate 16,31,128,65,"Print" GroupBox 8,8,64,52,"Amplifier Output",.Junk OptionGroup .Orientation OptionButton 16,20,51,8,"10 Watts",.Ten OptionButton 16,32,51,8,"50 Watts",.Fifty OptionButton 16,44,51,8,"100 Watts",.Hundred OKButton 80,8,40,14 End Dialog Dim PowerDialog As PowerTemplate Dialog PowerDialog End Sub </pre>
See Also	<p>CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin (on page 348) Dialog (on page 348) (statement), PictureButton (on page 659) (statement).</p>
Note	<p>Accelerators are underlined, and the accelerator combination Alt+letter is used.</p>

OptionGroup (statement)

Syntax	<p>OptionGroup .Identifier</p>
Description	<p>Specifies the start of a group of option buttons within a dialog box template.</p>
Comments	<p>The .Identifier parameter specifies the name by which the group of option buttons can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates an integer variable whose value corresponds to the index of the selected option button within the group (0 is the first option button, 1 is the second option button, and so on). This variable can be accessed using the following syntax: DialogVariable.Identifier. This statement can only appear within a dialog box template (that is, between the Begin Dialog and End</p>

	<p>Dialog statements). When the dialog box is created, the option button specified by <code>.Identifier</code> will be on; all other option buttons in the group will be off. When the dialog box is dismissed, the <code>.Identifier</code> will contain the selected option button.</p>
Example	<p>This example creates a group of option buttons.</p> <pre> Sub Main() Begin Dialog PowerTemplate 16,31,128,65,"Print" GroupBox 8,8,64,52,"Amplifier Output",.Junk OptionGroup .Orientation OptionButton 16,20,51,8,"10 Watts",.Ten OptionButton 16,32,51,8,"50 Watts",.Fifty OptionButton 16,44,51,8,"100 Watts",.Hundred OKButton 80,8,40,14 End Dialog Dim PowerDialog As PowerTemplate Dialog PowerDialog End Sub </pre>
See Also	<p>CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin Dialog (on page 348) (statement), PictureButton (on page 659) (statement).</p>

Or (operator)

Syntax	expression1 Or expression2		
Description	Performs a logical or binary disjunction on two expressions.		
Comments	If both expressions are either Boolean , Boolean variants, or Null variants, then a logical disjunction is performed as follows:		
	If the first expression is	and the second expression is	then: the result is

	TRUE	TRUE	TRUE
	TRUE	FALSE	TRUE
	TRUE	NULL	TRUE
	FALSE	TRUE	TRUE
	FALSE	FALSE	FALSE
	FALSE	NULL	NULL
	NULL	TRUE	TRUE
	NULL	FALSE	NULL
	NULL	NULL	NULL

Binary Disjunction If the two expressions are **Integer**, then a binary disjunction is performed, returning an **Integer** result. All other numeric types (including **Empty** variants) are converted to **Long** and a binary disjunction is then performed, returning a **Long** result. Binary disjunction forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:

1	Or	1	=	1	Example
0	Or	1	=	1	5 10101001
1	Or	0	=	1	6 01101010
0	Or	0	=	0	Or 11101011

Example 1 This first example shows the use of logical Or.

```

Sub Main()

    temperature_alert = True

    pressure_alert = False

    If temperature_alert Or pressure_alert Then

        MsgBox "You had better run!", vbExclamation, "Nuclear Disaster Imminent"

    End If

End Sub
    
```

Example 2 This second example shows the use of binary Or.

```

Sub Main()

    Dim w As Integer

    TryAgain:
    
```


	<pre> s\$ = InputBox("Enter a hex number (four digits max).", "Binary Or Example") If Mid(s\$,1,1) <> "&" Then s\$ = "&H" & s\$ End If If Not IsNumeric(s\$) Then Goto TryAgain w = Cint(s\$) MsgBox "Your number is &H" & Hex(w) w = w Or &H8000 MsgBox "Your number with the high bit set is &H" & Hex(w) End Sub </pre>
See	Operator Precedence (on page 648) (topic); Xor (on page 795) (operator); Eqv (on page 489) (operator); Imp (on page 557) (operator); And (on page 309) (operator).
Also	

P

P

Pi (constant)
Picture (statement)
PictureButton (statement)
Pmt (function)
PointSetMultiple (function)
PointSetMultipleEX (function)
PopupMenu (function)
PPmt (function)
Print (statement)
Print# (statement)
Private (statement)
Public (statement)
PushButton (statement)

Put (statement)
Pv (function)

Pi (constant)

Syntax	Pi
Description	The Double value 3.141592653589793238462643383279 .
Comments	<p>Pi can also be determined using the following formula:</p> <pre>4 * Atn(1)</pre>
Example	<p>This example illustrates the use of the Pi constant.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() dia = InputBox("Enter a circle diameter to compute. ","Compute Circle") circ# = Pi * dia area# = Pi * ((dia / 2) ^ 2) msg1 = "Diameter: " & dia & crlf msg1 = msg1 & "Circumference: " & Format(circ#,"Standard") & crlf msg1 = msg1 & "Area: " & Format(area#,"Standard") MsgBox msg1 End Sub</pre>
See Also	Tan (on page 751) (function); Atn (on page 337) (function); Cos (on page 385) (function); Sin (on page 718) (function).

Picture (statement)

Syntax	Picture X,Y,width,height,PictureName\$,PictureType [,Identifier] [,style]
Description	Creates a picture control in a dialog box template.
Comments	Picture controls are used for the display of graphics images only. The user cannot interact with these controls. The Picture statement accepts the following parameters:

	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	PicName\$	String containing the name of the picture. If PictureType is 0, then this name specifies the name of the file containing the image. If PictureType is 10, then PictureName\$ specifies the name of the image within the resource of the picture library. If PictureName\$ is empty, then no picture will be associated with the control. A picture can later be placed into the picture control using the DlgSetPicture statement.
	Picture-Type	Integer specifying the source for the image. The following sources are supported:
	0	The image is contained in a file on disk.
	10	The image is contained in a picture library as specified by the PicName\$ parameter on the Begin Dialog statement.
	.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). If omitted, then the first two words of PictureName\$ are used.
	style	Specifies whether the picture is drawn within a 3D frame. It can be any of the following values:
	0	Draw the picture control with a normal frame.
	1	Draw the picture control with a 3D frame.
		If omitted, then the picture control is drawn with a normal frame.
		The picture control extracts the actual image from either a disk file or a picture library. In the case of bitmaps, both 2- and 16-color bitmaps are supported. In the case of WMFs, the Basic Control Engine supports the Placeable Windows Metafile. If PictureName\$ is a zero-length string, then the picture is removed from the picture control, freeing any memory associated with that picture.
Example 1		This first example shows how to use a picture from a file.

	<pre> Sub Main() Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction" OKButton 240,8,40,14 Picture 8,8,224,64,"c:\bitmaps\logo.bmp",0,.Logo End Dialog Dim LogoDialog As LogoDialogTemplate Dialog LogoDialog End Sub </pre>
<p>Example 2</p>	<p>This second example shows how to use a picture from a picture library with a 3D frame.</p> <pre> Sub Main() Begin Dialog LogoDialogTemplate 16,31,288,76,"Introduction",,"pictures.dll" OKButton 240,8,40,14 Picture 8,8,224,64,"CompanyLogo",10,.Logo,1 End Dialog Dim LogoDialog As LogoDialogTemplate Dialog LogoDialog End Sub </pre>
<p>See Also</p>	<p>CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin (on page 348) Dialog (on page 348) (statement), PictureButton (on page 659) (statement); DlgSetPicture (on page 443) (statement).</p>
<p>Notes</p>	<p>Picture controls can contain either a bitmap or a WMF (Windows metafile). When extracting images from a picture library, the Basic Control Engine assumes that the resource type for metafiles is 256. Picture libraries are implemented as DLLs on the Windows and Win32 platforms.</p>

PictureButton (statement)

<p>Syntax</p>	<p>PictureButton X,Y,width,height,PictureName\$,PictureType [,Identifier]</p>
----------------------	--

De- scrip- tion	Creates a picture button control in a dialog box template.	
Com- ments	Picture button controls behave very much like a push button controls. Visually, picture buttons are different than push buttons in that they contain a graphic image imported either from a file or from a picture library. The PictureButton statement accepts the following parameters:	
	Para- meter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	Pic- ture- Name\$	String containing the name of the picture. If PictureType is 0, then this name specifies the name of the file containing the image. If PictureType is 10, then PictureName\$ specifies the name of the image within the resource of the picture library. If PictureName\$ is empty, then no picture will be associated with the control. A picture can later be placed into the picture control using the DlgSetPicture statement.
	Pic- ture- Type	Integer specifying the source for the image. The following sources are supported:
		0 The image is contained in a file on disk.
		10 The image is contained in a picture library as specified by the PicName\$ parameter on the Begin Dialog statement.
	.Iden- tifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).
	The picture button control extracts the actual image from either a disk file or a picture library, depending on the value of PictureType. The supported picture formats vary from platform to platform. If PictureName\$ is a zero-length string, then the picture is removed from the picture button control, freeing any memory associated with that picture.	
Exam- ple 1	This first example shows how to use a picture from a file. <pre>Sub Main() Begin Dialog LogoDialogTemplate 16,32,288,76,"Introduction"</pre>	

	<pre> OKButton 240,8,40,14 PictureButton 8,4,224,64,"c:\bitmaps\logo.bmp",0,.Logo End Dialog Dim LogoDialog As LogoDialogTemplate Dialog LogoDialog End Sub </pre>
Example 2	<p>This second example shows how to use a picture from a picture library.</p> <pre> Sub Main() Begin Dialog LogoDialogTemplate 16,31,288,76,"Introduction",,"pictures.dll" OKButton 240,8,40,14 PictureButton 8,4,224,64,"CompanyLogo",10,.Logo End Dialog Dim LogoDialog As LogoDialogTemplate Dialog LogoDialog End Sub </pre>
See Also	<p>CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin (on page 348) Dialog (on page 348) (statement), Picture (on page 657) (statement); DlgSetPicture (on page 443) (statement).</p>
Notes	<p>Picture controls can contain either a bitmap or a WMF (Windows metafile). When extracting images from a picture library, the Basic Control Engine assumes that the resource type for metafiles is 256. Picture libraries are implemented as DLLs on the Win32 platforms. Picture controls can contain either bitmaps or Windows metafiles.</p>

Pmt (function)

Syntax	Pmt (Rate,NPer,Pv,Fv,Due)
Description	Returns the payment for an annuity based on periodic fixed payments and a constant rate of interest.

Com-ments	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans. The Pmt function requires the following parameters:	
	Pa- ra- me- ter	Description
	Rate	Double representing the interest rate per period. If the periods are given in months, be sure to normalize annual rates by dividing them by 12.
	NPer	Double representing the total number of payments in the annuity.
	Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
	Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be 0.
	Due	Integer indicating when payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.
	Rate and NPer must be expressed in the same units. If Rate is expressed in months, then NPer must also be expressed in months. Positive numbers represent cash received, whereas negative numbers represent cash paid out.	
Exam- ple	This example calculates the payment necessary to repay a \$1,000.00 loan over 36 months at an annual rate of 10%. Payments are due at the beginning of the period.	
	<pre> Sub Main() x = Pmt((.1/12),36,1000.00,0,1) msg1 = "The payment to amortize \$1,000 over 36 months @ 10% is: " MsgBox msg1 & Format(x,"Currency") End Sub </pre>	
See Also	IPmt (on page 566) (function); NPer (on page 629) (function); PPmt (on page 663) (function); Rate (on page 680) (function).	

PopupMenu (function)

Syn- tax	PopupMenu (MenuItems\$())
De- scrip- tion	Displays a pop-up menu containing the specified items, returning an Integer representing the index of the selected item.
Com- ments	If no item is selected (that is, the pop-up menu is canceled), then a value of 1 less than the lower bound is returned (normally, -1). This function creates a pop-up menu using the string elements in the given array. Each array element is used as a menu item. A zero-length string results in a separator bar in the menu. The pop-up menu is created with the upper left corner at the current mouse position. A runtime error results if MenuItems\$ is not a single-dimension array. Only one pop-up menu can be displayed at a time. An error will result if another script executes this function while a pop-up menu is visible.
Exam- ple	<pre>Sub Main() Dim a\$() AppList a\$ w% = PopupMenu(a\$) End Sub</pre>
See Also	SelectBox (on page 709) (function).

PPmt (function)

Syn- tax	PPmt (Rate,Per,NPer,Pv,Fv,Due)				
De- scrip- tion	Calculates the principal payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate.				
Com- ments	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans. The PPmt function requires the following parameters:				
	<table border="1"> <thead> <tr> <th>Pa- ra- me- ter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Pa- ra- me- ter	Description		
Pa- ra- me- ter	Description				

	Rate	Double representing the interest rate per period.
	Per	Double representing the number of payment periods. Per can be no less than 1 and no greater than NPer.
	NPer	Double representing the total number of payments in your annuity.
	Pv	Double representing the present value of your annuity. In the case of a loan, the present value would be the amount of the loan.
	Fv	Double representing the future value of your annuity. In the case of a loan, the future value would be 0 .
	Due	Integer indicating when payments are due. If this parameter is 0 , then payments are due at the end of each period; if it is 1 , then payments are due at the start of each period.
		Rate and NPer must be in the same units to calculate correctly. If Rate is expressed in months, then NPer must also be expressed in months. Negative values represent payments paid out, whereas positive values represent payments received.
Example		<p>This example calculates the principal paid during each year on a loan of \$1,000.00 with an annual rate of 10% for a period of 10 years. The result is displayed as a table containing the following information: payment, principal payment, principal balance.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) pay = Pmt(.1,10,1000.00,0,1) msg1 = "Amortization table for 1,000" & crlf & "at 10% annually for" msg1 = msg1 & " 10 years: " & crlf & crlf bal = 1000.00 For per = 1 to 10 prn = PPmt(.1,per,10,1000,0,0) bal = bal + prn msg1 = msg1 & Format(pay,"Currency") & " " & Format\$(Prn,"Currency") msg1 = msg1 & " " & Format(bal,"Currency") & crlf Next per MsgBox msg1 End Sub </pre>
See Also		IPmt (on page 566) (function); NPer (on page 629) (function); PPmt (on page 663) (function); Rate (on page 680) (function).

Print (statement)

Syntax	Print [{ Spc (n) Tab (n)][expressionlist][{ ; ,}]	
Description	Prints data to an output device.	
Comments	The actual output device depends on the platform on which the Basic Control Engine is running. The following table describes how data of different types is written:	
	Data Type	Description
	String	Printed in its literal form, with no enclosing quotes.
	Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.
	Boolean	Printed as TRUE or FALSE.
	Date	Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is consistent with the "general date" format understood by the Format/Format\$ functions).
	Empty	Nothing is printed.
	Null	Prints NULL.
	User-defined errors	Printed as "Error code", where code is the value of the user-defined error. The word "Error" is not translated.
	<p>Each expression in expressionlist is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces. If the last expression in the list is not followed by a comma or a semicolon, then a carriage return is printed to the file. If the last expression ends with a semicolon, no carriage return is printed. The next Print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line. The Tab and Spc functions provide additional control over the column position. The Tab function moves the file position to the specified column, whereas the Spc function outputs the specified number of spaces.</p>	

Example	<pre> Sub Main() i% = 10 s\$ = "This is a test." Print "The value of i=";i%,"the value of s=";s\$ 'This example prints the value of i% in print zone 1 and s\$ in print 'zone 3. Print i%,,s\$ 'This example prints the value of i% and s\$ separated by 10 spaces. Print i%;Spc(10);s\$ 'This example prints the value of i in column 1 and s\$ in column 30. Print i%;Tab(30);s\$ 'This example prints the value of i% and s\$. Print i%;s\$, Print 67 End Sub </pre>
Note	<p>On Win32, the <code>Print</code> statement prints data to <code>stdout</code>.</p>

Print# (statement)

Syntax	<code>Print #filename, [[{Spc(n) Tab(n)}][expressionlist][{; ,}]]</code>								
Description	Writes data to a sequential disk file.								
Comments	The filename parameter is a number that is used by the Basic Control Engine to refer to the open file, the number passed to the <code>Open</code> statement. The following table describes how data of different types is written:								
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left; padding: 5px;">Data Type</th> <th style="text-align: left; padding: 5px;">Description</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">String</td> <td style="padding: 5px;">Printed in its literal form, with no enclosing quotes.</td> </tr> <tr> <td style="padding: 5px;">Any numeric type</td> <td style="padding: 5px;">Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.</td> </tr> <tr> <td style="padding: 5px;">Boolean</td> <td style="padding: 5px;">Printed as TRUE or FALSE.</td> </tr> </tbody> </table>	Data Type	Description	String	Printed in its literal form, with no enclosing quotes.	Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.	Boolean	Printed as TRUE or FALSE.
Data Type	Description								
String	Printed in its literal form, with no enclosing quotes.								
Any numeric type	Printed with an initial space reserved for the sign (space = positive). Additionally, there is a space following each number.								
Boolean	Printed as TRUE or FALSE.								

	<p>Date Printed using the short date format. If either the date or time component is missing, only the provided portion is printed (this is consistent with the "general date" format understood by the Format/Format\$ functions).</p>
	<p>Empty Nothing is printed.</p>
	<p>Null Prints NULL.</p>
<p>User-defined errors</p>	<p>Printed to files as "Error code", where code is the value of the user-defined error. The word "Error" is not translated.</p>
<p>Each expression in expressionlist is separated with either a comma (,) or a semicolon (;). A comma means that the next expression is output in the next print zone. A semicolon means that the next expression is output immediately after the current expression. Print zones are defined every 14 spaces. If the last expression in the list is not followed by a comma or a semicolon, then an end-of-line is printed to the file. If the last expression ends with a semicolon, no end-of-line is printed. The next Print statement will output information immediately following the expression. If the last expression in the list ends with a comma, the file pointer is positioned at the start of the next print zone on the current line.</p>	
<p>The Write statement always outputs information ending with an end-of-line. Thus, if a Print statement is followed by a Write statement, the file pointer is positioned on a new line. The Print statement can only be used with files that are opened in Output or Append mode. The Tab and Spc functions provide additional control over the file position. The Tab function moves the file position to the specified column, whereas the Spc function outputs the specified number of spaces. In order to correctly read the data using the Input# statement, you should write the data using the Write statement.</p>	
<p>Example</p>	<pre> Sub Main() 'This example opens a file and prints some data. Open "test.dat" For Output As #1 i% = 10 s\$ = "This is a test." Print #1,"The value of i=";i%,"the value of s=";s\$ 'This example prints the value of i% in print zone 1 and s\$ in 'print zone 3. Print #1,i%,,s\$ 'This example prints the value of i% and s\$ separated by ten spaces. Print #1,i%;Spc(10);s\$ 'This example prints the value of i in column 1 and s\$ in column 30. </pre>

	<pre> Print #1,i%;Tab(30);s\$ 'This example prints the value of i% and s\$. Print #1,i%;s\$, Print #1,67 Close #1 Kill "test.dat" End Sub </pre>
See	Open (on page 642) (statement); Put (on page 673) (statement); Write# (on page 793)
Also	(statement).
Note	The end-of-line character can be either the carriage-return/line-feed pair, or the line-feed character.

If you want it to go to a file you need the # otherwise it goes to standard out and uses the first variable (in this case f) as the first item to output to standard out. So that print line should be

Print #F, "This is a test"

Private (statement)

Syn- tax	Private name [(subscripts)] [As type] [,name [(subscripts)] [As type]]...
De- scrip- tion	Declares a list of private variables and their corresponding types and sizes.
Com- ments	Private variables are global to every Sub and Function within the currently executing script. If a type-declaration character is used when specifying name (such as %, @, &, \$, or !), the optional [As type] expression is not allowed. For example, the following are allowed: Private foo As Integer Private foo% The subscripts parameter allows the declaration of arrays. This parameter uses the following syntax: [lower To] upper [, [lower To] upper]... The lower and upper parameters are integers specifying the lower and upper bounds of the array. If lower is not specified, then the lower bound as specified by Option Base is used (or 1 if no Option Base statement has been encountered). Up to 60 array dimensions are allowed.
	The total size of an array (not counting space for strings) is limited to 64K. Dynamic arrays are declared by not specifying any bounds: Private a() The type parameter specifies the type of the data item being declared. It can be any of the following data types: String, Integer, Long, Sin-

gle, Double, Currency, Object, data object, built-in data type, or any user-defined data type. If a variable is seen that has not been explicitly declared with either **Dim**, **Public**, or **Private**, then it will be implicitly declared local to the routine in which it is used. Fixed-Length Strings Fixed-length strings are declared by adding a length to the **String** type-declaration character: Private name As String * length where length is a literal number specifying the string's length. Initial Values All declared variables are given initial values, as described in the following table:

Data Type	Description
Integer	0
Long	0
Double	0.0
Single	0.0
Currency	0.0
Object	Nothing
Date	December 31, 1899 00:00:00
Boolean	False
Variant	Empty
String	"" (zero-length string)
User-defined type	Each element of the structure is given a default value, as described above.
Arrays	Each element of the array is given a default value, as described above.

Example This example sets the value of variable x# in two separate routines to show the behavior of private variables.

```
Private x#
Sub Area()
    x# = 10 'Set this copy of x# to 10 and display
    MsgBox x#
End Sub
Sub Main()
    x# = 100 'Set this copy of x# to 100 and display after calling the Area subroutine
    Area
    MsgBox x#
End Sub
```

See	Dim (on page 426) (statement); Redim (on page 688) (statement); Public (on page 670)
Also	(statement); Option Base (on page 649) (statement).

Public (statement)

Syntax	Public name [(subscripts)] [As type] [,name [(subscripts)] [As type]]...										
Description	Declares a list of public variables and their corresponding types and sizes.										
Comments	Public variables are global to all Sub s and Function s in all scripts. If a type-declaration character is used when specifying name (such as %, @, &, \$, or !), the optional [As type] expression is not allowed. For example, the following are allowed: <code>Public foo As Integer</code> <code>Public foo%</code> The subscripts parameter allows the declaration of arrays. This parameter uses the following syntax: [lower To] upper [, [lower To] upper]... The lower and upper parameters are integers specifying the lower and upper bounds of the array. If lower is not specified, then the lower bound as specified by <code>Option Base</code> is used (or 1 if no <code>Option Base</code> statement has been encountered). Up to 60 array dimensions are allowed.										
	The total size of an array (not counting space for strings) is limited to 64K. Dynamic arrays are declared by not specifying any bounds: <code>Public a()</code> The type parameter specifies the type of the data item being declared. It can be any of the following data types: String , Integer , Long , Single , Double , Currency , Object , data object, built-in data type, or any user-defined data type. If a variable is seen that has not been explicitly declared with either Dim , Public , or Private , then it will be implicitly declared local to the routine in which it is used. For compatibility, the keyword Global is also supported. It has the same meaning as Public . Fixed-Length Strings Fixed-length strings are declared by adding a length to the String type-declaration character: <code>Public name As String * length</code> where length is a literal number specifying the string's length. Initial Values All declared variables are given initial values, as described in the following table:										
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Integer</td> <td>0</td> </tr> <tr> <td>Long</td> <td>0</td> </tr> <tr> <td>Double</td> <td>0.0</td> </tr> <tr> <td>Single</td> <td>0.0</td> </tr> </tbody> </table>	Parameter	Description	Integer	0	Long	0	Double	0.0	Single	0.0
Parameter	Description										
Integer	0										
Long	0										
Double	0.0										
Single	0.0										

	Currency	0.0
	Date	December 31, 1899 00:00:00
	Object	Nothing
	Boolean	False
	Variant	Empty
	String	"" (zero-length string)
	User-defined type	Each element of the structure is given a default value, as described above.
	Arrays	Each element of the array is given a default value, as described above.
	Sharing Variables When sharing variables, you must ensure that the declarations of the shared variables are the same in each script that uses those variables. If the public variable being shared is a user-defined structure, then the structure definitions must be exactly the same.	
Example	<p>This example uses a subroutine to calculate the area of ten circles and displays the result in a dialog box. The variables R and Ar are declared as Public variables so that they can be used in both Main and Area.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Public x#,ar# Sub Area() ar# = (x# ^ 2) * Pi End Sub Sub Main() msg1 = "The area of the ten circles are:" & crlf & crlf For x# = 1 To 10 Area msg1 = msg1 & x# & ": " & Format(ar#,"fixed") & Basic.Eoln\$ Next x# MsgBox msg1 End Sub </pre>	
See Also	Dim (on page 426) (statement); Redim (on page 688) (statement); Option Base (on page 649) (statement).	

PushButton (statement)

Syntax	<code>PushButton X,Y,width,height,title\$ [,.Identifier]</code>	
Description	Defines a push button within a dialog box template.	
Comments	Choosing a push button causes the dialog box to close (unless the dialog function redefines this behavior). This statement can only appear within a dialog box template (that is, between the <code>Begin Dialog</code> and <code>End Dialog</code> statements). The PushButton statement accepts the following parameters:	
	Parameter	Description
	X, Y	Integer coordinates specifying the position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer coordinates specifying the dimensions of the control in dialog units.
	title\$	String containing the text that appears within the push button. This text may contain an ampersand character to denote an accelerator letter, such as " &Save " for Save .
	.Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable).
	If a push button is the default button, it can be selected by pressing Enter on a nonbutton control. A dialog box template must contain at least one OKButton , CancelButton , or PushButton statement (otherwise, the dialog box cannot be dismissed).	
Example	<p>This example creates a bunch of push buttons and displays which button was pushed.</p> <pre> Sub Main() Begin Dialog ButtonTemplate 17,33,104,84,"Buttons" OKButton 8,4,40,14,.OK CancelButton 8,24,40,14,.Cancel PushButton 8,44,40,14,"1",.Button1 PushButton 8,64,40,14,"2",.Button2 PushButton 56,4,40,14,"3",.Button3 PushButton 56,24,40,14,"4",.Button4 PushButton 56,44,40,14,"5",.Button5 PushButton 56,64,40,14,"6",.Button6 </pre>	

	<pre> End Dialog Dim ButtonDialog As ButtonTemplate WhichButton% = Dialog(ButtonDialog) MsgBox "You pushed button " & WhichButton% End Sub </pre>
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); Text (on page 752) (statement); TextBox (on page 753) (statement); Begin (on page 348) Dialog (on page 348) (statement), PictureButton (on page 659) (statement); DlgSetPicture (on page 443) (statement).
Note	Accelerators are underlined, and the accelerator combination Alt+ letter is used.

Put (statement)

Syntax	Put [#] filename, [recordnumber], variable	
Description	Writes data from the specified variable to a Random or Binary file.	
Comments	The Put statement accepts the following parameters:	
	Parameter	Description
	filename	Integer representing the file to be written to. This is the same value as returned by the Open statement.
	recordnumber	Long specifying which record is to be written to the file. For Binary files, this number represents the first byte to be written starting with the beginning of the file (the first byte is 1). For Random files, this number represents the record number starting with the beginning of the file (the first record is 1). This value ranges from 1 to 2147483647. If the recordnumber parameter is omitted, the next record is written to the file (if no records have been written yet, then the first record in the file is written). When record-

		number is omitted, the commas must still appear, as in the following example: Put #1,,recvar If recordlength is specified, it overrides any previous change in file position specified with the Seek statement.
	The variable parameter is the name of any variable of any of the following types:	
	Variable Type	File Storage Description
	Integer	2 bytes are written to the file.
	Long	4 bytes are written to the file.
	String (variable-length)	In Binary files, variable-length strings are written by first determining the specified string variable's length, then writing that many bytes to the file. In Random files, variable-length strings are written by first writing a 2-byte length, then writing that many characters to the file.
	String (fixed-length)	Fixed-length strings are written to Random and Binary files in the same way: the number of characters equal to the string's declared length are written.
	Double	8 bytes are written to the file (IEEE format).
	Single	4 bytes are written to the file (IEEE format).
	Date	8 bytes are written to the file (IEEE double format).
	Boolean	2 bytes are written to the file (either -1 for TRUE or 0 for FALSE).
	Variant	A 2-byte VarType is written to the file followed by the data as described above. With variants of type 10 (user-defined errors), the 2-byte VarType is followed by a 2-byte unsigned integer (the error value), which is then followed by 2 additional bytes of information. The exception is with strings, which are always preceded by a 2-byte string length.
	User-defined types	Each member of a user-defined data type is written individually. In Binary files, variable-length strings within user-defined types are written by first writing a 2-byte length followed by the string's content. This storage is different than variable-length strings outside of user-defined types. When writing user-defined types, the record length must be greater than or equal to the combined size of each element within the data type.
	Arrays	Arrays cannot be written to a file using the Put statement.
	Objects	Object variables cannot be written to a file using the Put statement.

	<p>With Random files, a runtime error will occur if the length of the data being written exceeds the record length (specified as the reclen parameter with the Open statement). If the length of the data being written is less than the record length, the entire record is written along with padding (whatever data happens to be in the I/O buffer at that time). With Binary files, the data elements are written contiguously: they are never separated with padding.</p>
<p>Example</p>	<p>This example opens a file for random write, then writes ten records into the file with the values 10-50. Then the file is closed and reopened in random mode for read, and the records are read with the Get statement. The result is displayed in a dialog box.</p> <pre data-bbox="305 598 1421 1365"> Sub Main() Open "test.dat" For Random Access Write As #1 For x = 1 To 10 r% = x * 10 Put #1,x,r% Next x Close Open "test.dat" For Random Access Read As #1 For x = 1 To 10 Get #1,x,r% msg1 = "Record " & x & " is: " & r% & Basic.Eoln\$ Next x MsgBox msg1 Close Kill "test.dat" End Sub </pre>
<p>See Also</p>	<p>Open (on page 642) (statement); Put (on page 673) (statement); Write# (on page 793) (statement); Print# (on page 666) (statement).</p>

Pv (function)

<p>Syn- tax</p>	<p>Pv (Rate,NPer,Pmt,Fv,Due)</p>
<p>De- scrip- tion</p>	<p>Calculates the present value of an annuity based on future periodic fixed payments and a constant rate of interest.</p>

Comments	The Pv function requires the following parameters:	
	Parameter	Description
	Rate	Double representing the interest rate per period. When used with monthly payments, be sure to normalize annual percentage rates by dividing them by 12.
	NPer	Double representing the total number of payments in the annuity.
	Pmt	Double representing the amount of each payment per period.
	Fv	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be 0.
	Due	Integer indicating when the payments are due for each payment period. A 0 specifies payment at the end of each period, whereas a 1 specifies payment at the start of each period.
	Rate and NPer must be expressed in the same units. If Rate is expressed in months, then NPer must also be expressed in months. Positive numbers represent cash received, whereas negative numbers represent cash paid out.	
Example	<p>This example demonstrates the present value (the amount you'd have to pay now) for a \$100,000 annuity that pays an annual income of \$5,000 over 20 years at an annual interest rate of 10%.</p> <pre> Sub Main() pval = Pv(.1,20,-5000,100000,1) MsgBox "The present value is: " & Format(pval,"Currency") End Sub </pre>	
See Also	Fv (on page 533) (function); IRR (on page 568) (function); MIRR (on page 606) (function); Npv (on page 630) (function).	

Q

QueEmpty (statement)

Syntax	QueEmpty
--------	----------

Description	Empties the current event queue.
Comments	After this statement, QueFlush will do nothing.
Example	<pre> 'This code begins a new queue, then drags a selection over a 'range of characters in Notepad. Sub Main() AppActivate "Notepad" QueEmpty 'Make sure the queue is empty. QueMouseDown ebLeftButton,1440,1393 QueMouseUp ebLeftButton,4147,2363 QueFlush True End Sub </pre>

R

R

Random (function)
Randomize (statement)
Rate (function)
RCPDownload (statement)
RCPDownloadEx (function)
RCPGroupExport (statement)
RCPGroupExportEx (function)
RCPGroupImport (statement)
RCPGroupImportEx (function)
RCPUpload (statement)
RCPUploadEx (function)
ReadIn\$ (function)
ReadInSection (statement)

Redim (statement)
Rem (statement)
Reset (statement)
Resume (statement)
Return (statement)
Right, Right\$, RightB, RightB\$ (functions)
Rmdir (statement)
Rnd (function)
RSet (statement)
RTrim, RTrim\$ (function)

Random (function)

Syntax	Random (min,max)
Description	Returns a Long value greater than or equal to min and less than or equal to max.
Comments	Both the min and max parameters are rounded to Long . A runtime error is generated if min is greater than max.
Example	<p>This example sets the randomize seed then generates six random numbers between 1 and 54 for the lottery.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a%(5) Randomize For x = 0 To 5 temp = Random(1,54) 'Eliminate duplicate numbers. For y = 0 To 5 If a(y) = temp Then found = true Next If found = false Then a(x) = temp Else x = x - 1 found = false </pre>

	<pre> Next ArraySort a msg1 = "" For x = 0 To 5 msg1 = msg1 & a(x) & crlf Next x MsgBox "Today's winning lottery numbers are: " & crlf & crlf & msg1 End Sub </pre>
See Also	Randomize (on page 679) (statement); Rnd (on page 694) (function).

Randomize (statement)

Syntax	Randomize [seed]
Description	Initializes the random number generator with a new seed.
Comments	If seed is not specified, then the current value of the system clock is used.
Example	<p>This example sets the randomize seed then generates six random numbers between 1 and 54 for the lottery.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a%(5) Randomize 'This sets the random seed. 'Omitting this line will cause the random numbers to be 'identical each time the sample is run. For x = 0 To 5 temp = Rnd(1) * 54 + 1 'Eliminate duplicate numbers. For y = 0 To 5 If a(y) = temp Then found = true Next If found = false Then a(x) = temp Else x = x - 1 found = false Next ArraySort a </pre>


```

msg1 = ""

For x = 0 To 5

    msg1 = msg1 & a(x) & crlf

Next x

MsgBox "Today's winning lottery numbers are: " & crlf & crlf & msg1

End Sub

```

See Also [Random \(on page 678\)](#) (function); [Rnd \(on page 694\)](#) (function).

Rate (function)

Syn- tax	Rate (NPer,Pmt,Pv,Fv,Due,Guess)	
De- scrip- tion	Returns the rate of interest for each period of an annuity.	
Com- ments	An annuity is a series of fixed payments made to an insurance company or other investment company over a period of time. Examples of annuities are mortgages and monthly savings plans. The Rate function requires the following parameters:	
	Parameter	Description
	NPer	Double representing the total number of payments in the annuity.
	Pmt	Double representing the amount of each payment per period.
	Pv	Double representing the present value of your annuity. In a loan situation, the present value would be the amount of the loan.
	Fv	Double representing the future value of the annuity after the last payment has been made. In the case of a loan, the future value would be zero.
	Due	Integer specifying when the payments are due for each payment period. A 0 indicates payment at the end of each period, whereas a 1 indicates payment at the start of each period.
	Guess	Double specifying a guess as to the value the Rate function will return. The most common guess is .1 (10 percent).
	Positive numbers represent cash received, whereas negative values represent cash paid out. The value of Rate is found by iteration. It starts with the value of Guess and cycles through the	

	calculation adjusting Guess until the result is accurate within 0.00001 percent. After 20 tries, if a result cannot be found, Rate fails, and the user must pick a better guess.
Example	<p>This example calculates the rate of interest necessary to save \$8,000 by paying \$200 each year for 48 years. The guess rate is 10%.</p> <pre> Sub Main() r# = Rate(48,-200,8000,0,1,.1) MsgBox "The rate required is: " & Format(r#,"Percent") End Sub </pre>
See Also	IPmt (on page 566) (function); NPer (on page 629) (function); Pmt (on page 661) (function); PPmt (on page 663) (function).

RCPDownload (statement)

Syntax	RCPDownload filename\$[, [recipeName\$][, [mapName\$][, [pointorval\$][, ispoint]]]]	
Description	Downloads the specified Recipe from the specified Recipe Group using the specified Map.	
Comments	The RCPDownload function takes the following parameters:	
	Parameter	Description
	filename\$	Required string containing the name of the Recipe Group file where the Recipe is located. The Recipe Group file must exist
	recipeName\$	Required string containing the name of the Recipe to be Downloaded.
	mapName\$	Required string containing the name of the Map to be used when Downloading the Recipe.
	pointorval\$	Required string containing the Batch Point or Batch ID to be used when Downloading the Recipe.
	ispoint	Required integer, set to 1 if pointorval\$ is a Batch Point.
Example	<pre>RCPDownload "D:\Bread.rgp", "White", "Line1", "Batch of White Bread", 0</pre>	

RCPDownloadEx (function)

Syntax	RCPDownloadEx (filename\$[, [recipename\$][, [mapname\$][, [pointorval\$][, ispoint]]])	
Description	Downloads the specified Recipe from the specified Recipe Group using the specified Map.	
Comments	The RCPDownloadEx function takes the following parameters:	
	Parameter	Description
	filename\$	Required string containing the name of the Recipe Group file where the Recipe is located. The Recipe Group file must exist
	recipename\$	Required string containing the name of the Recipe to be Downloaded.
	mapname\$	Required string containing the name of the Map to be used when Downloading the Recipe.
	pointorval\$	Required string containing the Batch Point or Batch ID to be used when Downloading the Recipe.
	ispoint	Required integer, set to 1 if pointorval\$ is a Batch Point.
	The value returned is one of the following constants.	
	RCP_SUCCESS	Function was successful.
	RCP_PT_UNAVAIL	Point in recipe was disabled or does not exist.
	RCP_NO_PERMISSION	Current user has no setpoint permissions for points in recipe.
	RCP_VAL_OUTSIDE_RANGE	Value to be written to point in recipe is outside setpoint range.
	RCP_FILER_ERR	Path or file not found
	RCP_OLE_ERR	OLE error in execution.
	RCP_UNKNOWN	Error not covered in one of the above.

Exam- ple	<code>RCPDownloadEx ("D:\Bread.rgp", "White", "Line1", "Batch of White Bread", 0)</code>
--------------	--

RCPGroupExport (statement)

Syntax	RCPGroupExport groupname\$, filename\$]	
Description	Exports the specified Recipe Group to a CSV file.	
Comments	The RCPGroupExport function takes the following parameters:	
	Parameter	Description
	group-name\$	Required string containing the name of the Recipe Group file. The Recipe Group file must exist.
	filename\$	Optional string containing the name of the CSV file.
Example	<code>RCPGroupExport "D:\Bread.rgp"</code>	

RCPGroupExportEx (function)

Syntax	RCPGroupExportEx (groupname\$, filename\$)	
Description	Exports the specified Recipe Group to a CSV file.	
Comments	The RCPGroupExportEx function takes the following parameters:	
	Parameter	Description
	groupname\$	Required string containing the name of the Recipe Group file. The Recipe Group file must exist.
	filename\$	Optional string containing the name of the CSV file.
	The value returned is one of these constants.	
	RCP_SUCCESS	Function was successful.
	RCP_PT_UNAVAIL	Point in recipe was disabled or does not exist.

	RCP_NO_PERMISSION	Current user has no setpoint permissions for points in recipe.
	RCP_VAL_OUTSIDE_RANGE	Value to be written to point in recipe is outside setpoint range.
	RCP_FILER_ERR	Path or file not found.
	RCP_OLE_ERR	OLE error in execution.
	RCP_UNKNOWN	Error not covered in one of the above.
Example	<pre>RCPGroupExportEx ("D:\Bread.rgp")</pre>	

RCPGroupImport (statement)

Syntax	RCPGroupImport groupname\$[, filename\$]	
Description	Imports the specified Recipe Group from a CSV file.	
Comments	The RCPGroupImport function takes the following parameters:	
	Parameter	Description
	group-name\$	Required string containing the name of the Recipe Group file.
	filename\$	Optional string containing the name of the CSV file.
Example	<pre>RCPGroupImport "D:\Bread.rgp", "Bread2.csv"</pre>	

RCPGroupImportEx (function)

Syntax	RCPGroupImportEx (groupname\$[, filename\$])	
Description	Imports the specified Recipe Group from a CSV file.	
Comments	The RCPGroupImportEx function takes the following parameters:	
	Parameter	Description
	groupname\$	Required string containing the name of the Recipe Group file.
	filename\$	Optional string containing the name of the CSV file.

	The value returned is one of these constants.	
	RCP_SUCCESS	Function was successful.
	RCP_PT_UNAVAIL	Point in recipe was disabled or does not exist.
	RCP_NO_PERMISSION	Current user has no setpoint permissions for points in recipe.
	RCP_VAL_OUTSIDE_RANGE	Value to be written to point in recipe is outside setpoint range.
	RCP_FILER_ERR	Path or file not found.
	RCP_OLE_ERR	OLE error in execution.
	RCP_UNKNOWN	Error not covered in one of the above.
Example	<pre>RCPGroupImportEx ("D:\Bread.rgp", "Bread2.csv")</pre>	

RCPUpload (statement)

Syntax	RCPUpload filename\$[[recipename\$][[mapname\$] [newname\$]]]	
Description	Uploads the specified Recipe to the specified Recipe Group using the specified Map.	
Comments	The RCPUpload function takes the following parameters:	
	Parameter	Description
	filename\$	Required string containing the name of the Recipe Group file where the Recipe is located. The Recipe Group file must exist.
	recipename\$	Optional string containing the name of the Recipe to be Uploaded.
	mapname\$	Optional string containing the name of the Map to be used when Uploading the Recipe.
	newname\$	Optional string containing the name of the Recipe to be uploaded. You may use a new or existing Recipe name.
Example	<pre>RCPUpload "D:\Bread.rgp", "White", "Line1", "NewWhite"</pre>	

RCPUploadEx (function)

Syntax	RCPUploadEx (filename\$[, [recipename\$] [, [mapname\$] [, newname\$]]])	
Description	Uploads the specified Recipe to the specified Recipe Group using the specified Map.	
Comments	The RCPUploadEx function takes the following parameters:	
	Parameter	Description
	filename\$	Required string containing the name of the Recipe Group file where the Recipe is located. The Recipe Group file must exist.
	recipename\$	Optional string containing the name of the Recipe to be Uploaded.
	mapname\$	Optional string containing the name of the Map to be used when Uploading the Recipe.
	newname\$	Optional string containing the name of the Recipe to be uploaded. You may use a new or existing Recipe name.
	The value returned is one of the following constants.	
	RCP_SUCCESS	Function was successful.
	RCP_PT_UN-AVAIL	Point in recipe was disabled or does not exist.
	RCP_NO_PER-MISSION	Current user has no setpoint permissions for points in recipe.
	RCP_VAL_OUT-SIDE_RANGE	Value to be written to point in recipe is outside setpoint range.
	RCP_FILER_ERR	Path or file not found
	RCP_OLE_ERR	OLE error in execution.
	RCP_UNKNOWN	Error not covered in one of the above.
Example	<pre>RCPUploadEx ("D:\Bread.rgp", "White", "Line1", "NewWhite")</pre>	

ReadIni\$ (function)

Syn-tax	ReadIni\$ (section\$,item\$[,filename\$])	
De-scrip-tion	Returns a String containing the specified item from an ini file.	
Com-ments	The <code>ReadIni\$</code> function takes the following parameters:	
	Para-meter	Description
	Sec-tion\$	String specifying the section that contains the desired variable, such as "windows". Section names are specified without the enclosing brackets.
	Item\$	String specifying the item whose value is to be retrieved.
	File-name\$	String containing the name of the ini file to read.
See Also	WriteIni (on page 795) (statement); ReadIniSection (on page 687) (statement)	
Notes	If the name of the ini file is not specified, then win.ini is assumed. If the filename\$ parameter does not include a path, then this statement looks for ini files in the Windows directory.	

ReadIniSection (statement)

Syn-tax	ReadIniSection section\$,ArrayOfItems()[,filename\$]	
De-scrip-tion	Fills an array with the item names from a given section of the specified ini file.	
Com-ments	The ReadIniSection statement takes the following parameters:	
	Para-meter	Description

Section\$	String specifying the section that contains the desired variables, such as "windows". Section names are specified without the enclosing brackets.
ArrayOfItems()	Specifies either a zero- or a one-dimensional array of strings or variants. The array can be either dynamic or fixed. If ArrayOfItems() is dynamic, then it will be redimensioned to exactly hold the new number of elements. If there are no elements, then the array will be redimensioned to contain no dimensions. You can use the <code>LBound</code> , <code>UBound</code> , and <code>ArrayDims</code> functions to determine the number and size of the new array's dimensions. If the array is fixed, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are initialized to zero-length strings (for String arrays) or Empty (for Variant arrays). A runtime error results if the array is too small to hold the new elements.
Filename\$	String containing the name of an ini file.
	On return, the ArrayOfItems() parameter will contain one array element for each variable in the specified ini section.
Example	<pre>Sub Main() Dim items() As String ReadIniSection "Windows", items\$ r% = SelectBox("INI Items", , items\$) End Sub</pre>
See Also	ReadIni\$ (on page 687) (function); WriteIni (on page 795) (statement)
Note	If the name of the ini file is not specified, then win.ini is assumed. If the filename\$ parameter does not include a path, then this statement looks for .ini files in the Windows directory.

Redim (statement)

Syntax	Redim [Preserve] variablename (subscriptRange) [As type],...
Description	Redimensions an array, specifying a new upper and lower bound for each dimension of the array.
Comments	The variablename parameter specifies the name of an existing array (previously declared using the Dim statement) or the name of a new array variable. If the array variable already ex-

ists, then it must previously have been declared with the **Dim** statement with no dimensions, as shown in the following example:

```
Dim a$() 'Dynamic array of strings (no dimensions yet)
```

Dynamic arrays can be redimensioned any number of times. The subscriptRange parameter specifies the new upper and lower bounds for each dimension of the array using the following syntax:

```
[lower To] upper [, [lower To] upper]...
```

If lower is not specified, then **0** is used (or the value set using the **Option Base** statement). A runtime error is generated if lower is less than upper. Array dimensions must be within the following range:

```
-32768 <= lower <= upper <= 32767
```

The type parameter can be used to specify the array element type. Arrays can be declared using any fundamental data type, user-defined data types, and objects. Re-dimensioning an array erases all elements of that array unless the **Preserve** keyword is specified. When this keyword is specified, existing data in the array is preserved where possible. If the number of elements in an array dimension is increased, the new elements are initialized to **0** (or empty string). If the number of elements in an array dimension is decreased, then the extra elements will be deleted. If the **Preserve** keyword is specified, then the number of dimensions of the array being re-dimensioned must either be zero or the same as the new number of dimensions.

Example This example uses the FileList statement to re-dim an array and fill it with filename strings. A new array is then re-dimmed to hold the number of elements found by FileList, and the FileList array is copied into it and partially displayed.

```
Sub Main()
    Dim fl$()
    FileList fl$,"*.*"
    count = Ubound(fl$)
    Redim nl$(Lbound(fl$) To Ubound(fl$))
    For x = 1 to count
        nl$(x) = fl(x)
    Next x
    MsgBox "The last element of the new array is: " & nl$(count)
End Sub
```

See	Dim (on page 426) (statement); Public (on page 670) (statement); Private (on page 668)
Also	(statement); ArrayDims (on page 328) (function); LBound (on page 581) (function); UBound (on page 765) (function).

Rem (statement)

Syntax	Rem text
Description	Causes the compiler to skip all characters on that line.
Example	<pre>Sub Main() Rem This is a line of comments that serves to illustrate the Rem workings of the code. You can insert comments to make it more Rem readable and maintainable in the future. End Sub</pre>
See Also	' (on page 291) (keyword); Comments (on page 390) (topic).

Reset (statement)

Syntax	Reset
Description	Closes all open files, writing out all I/O buffers.
Example	<p>This example opens a file for output, closes it with the Reset statement, then deletes it with the Kill statement.</p> <pre>Sub Main() Open "test.dat" for Output Access Write as # 1 Reset Kill "test.dat" If FileExists("test.dat") Then MsgBox "The file was not deleted." Else MsgBox "The file was deleted." End If End Sub</pre>

See Also [Close \(on page 373\)](#) (statement); [Open \(on page 642\)](#) (statement).

Resume (statement)

Syn- tax	Resume {[0] Next label}
De- scrip- tion	Ends an error handler and continues execution.
Com- ments	The form Resume 0 (or simply Resume by itself) causes execution to continue with the statement that caused the error. The form Resume Next causes execution to continue with the statement following the statement that caused the error.
	The form Resume label causes execution to continue at the specified label. The Resume statement resets the error state. This means that, after executing this statement, new errors can be generated and trapped as normal.
Exam- ple	<p>This example accepts two integers from the user and attempts to multiply the numbers together. If either number is larger than an integer, the program processes an error routine and then continues program execution at a specific section using 'Resume <label>'. Another error trap is then set using 'Resume Next'. The new error trap will clear any previous error branching and also 'tell' the program to continue execution of the program even if an error is encountered.</p> <pre> Sub Main() Dim a%,b%,x% Again: On Error Goto Overflow a% = InputBox("Enter 1st integer to multiply","Enter Number") b% = InputBox("Enter 2nd integer to multiply","Enter Number") On Error Resume Next 'Continue program execution at next line x% = a% * b% 'if an error (integer overflow) occurs. If err = 0 Then MsgBox a% & " * " & b% & " = " & x% Else MsgBox a% & " * " & b% & " cause an integer overflow!" End If Exit Sub </pre>

	<pre> Overflow: 'Error handler. MsgBox "You've entered a non-integer value, try again!" Resume Again End Sub </pre>
See Also	Error Handling (on page 495) (topic); On Error (on page 640) (statement).

Return (statement)

Syntax	Return
Description	Transfers execution control to the statement following the most recent GoSub .
Comments	A runtime error results if a Return statement is encountered without a corresponding GoSub statement.
Example	<p>This example calls a subroutine and then returns execution to the Main routine by the Return statement.</p> <pre> Sub Main() GoSub SubTrue MsgBox "The Main routine continues here." Exit Sub SubTrue: MsgBox "This message is generated in the subroutine." Return Exit Sub End Sub </pre>
See Also	GoSub (on page 543) (statement).

Right, Right\$, RightB, RightB\$ (functions)

Syntax	<code>Right[\$](string, length) RightB[\$](string, length)</code>	
Description	Functions return the rightmost length for the following.	
	Function	Returns rightmost length for;

	<code>Right</code> and <code>RightB</code>	Characters
	<code>RightB</code> and <code>RightB\$</code>	Bytes
Comments	Functions return the following.	
	Functions	Return
	<code>Right</code> and <code>RightB</code>	String variant
	<code>Right\$</code> and <code>RightB\$</code>	String
	These functions take the following named parameters:	
	Parameter	Description
	string	String from which characters are returned. A runtime error is generated if string is NULL.
	length	Integer specifying the number of characters or bytes to return as follows.
		Length is
		Returns
		0
		Zero-length string is returned.
		Greater than or equal to the length of the string
		String is returned.
		The <code>RightB</code> and <code>RightB\$</code> functions are used to return byte data from strings containing byte data.
Example	<pre> 'This example shows the Right\$ function used in a routine to 'change uppercase names to lowercase with an uppercase first 'letter. Sub Main() lname\$ = "WILLIAMS" x = Len(lname\$) rest\$ = Right\$(lname\$,x - 1) fl\$ = Left\$(lname\$,1) lname\$ = fl\$ & LCase\$(rest\$) MsgBox "The converted name is: " & lname\$ End Sub </pre>	

See Also	Left , Left\$, LeftB , LeftB\$ (<i>on page 582</i>) (functions)
----------	--

RmDir (statement)

Syntax	RmDir dir\$
Comments	Removes the directory specified by the String contained in dir\$.
Example	<p>This routine creates a directory and then deletes it with RmDir.</p> <pre> Sub Main() On Error Goto ErrMake MkDir("test01") On Error Goto ErrRemove RmDir("test01") ErrMake: MsgBox "The directory could not be created." Exit Sub ErrRemove: MsgBox "The directory could not be removed." Exit Sub End Sub </pre>
See Also	ChDir (<i>on page 359</i>) (statement); ChDrive (<i>on page 359</i>) (statement); CurDir , CurDir\$ (<i>on page 387</i>) (functions); Dir , Dir\$ (<i>on page 427</i>) (functions); MkDir (<i>on page 608</i>) (statement).

Rnd (function)

Syntax	Rnd [(number)]		
Description	Returns a random Single number between 0 and 1.		
Comments	If number is omitted, the next random number is returned. Otherwise, the number parameter has the following meaning:		
	<table border="1" style="width: 100%;"> <tr> <td style="width: 50%;">If</td> <td>Then</td> </tr> </table>	If	Then
If	Then		

	Number < 0	Always returns the same number.
	Number = 0	Returns the last number generated.
	Number > 0	Returns the next random number.
Example	<p>This example sets the randomize seed then generates six random numbers between 1 and 54 for the lottery.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a%(5) Randomize For x = 0 To 5 temp = Rnd(1) * 54 + 1 Eliminate duplicate numbers. or y = 0 To 5 If a(y) = temp Then found = true Next If found = false Then a(x) = temp Else x = x - 1 found = false Next ArraySort a msg1 = "" For x = 0 To 5 msg1 = msg1 & a(x) & crlf Next x MsgBox "Today's winning lottery numbers are: " & crlf & crlf & msg1 End Sub </pre>	
See Also	Randomize (on page 679) (statement); Random (on page 678) (function)	

RSet (statement)

Syntax	RSet destvariable = source
--------	-----------------------------------

De- scrip- tion	Copies the source string source into the destination string destvariable.
Com- ments	If source is shorter in length than destvariable, then the string is right-aligned within destvariable and the remaining characters are padded with spaces. If source is longer in length than destvariable, then source is truncated, copying only the leftmost number of characters that will fit in destvariable. A runtime error is generated if source is Null . The destvariable parameter specifies a String or Variant variable. If destvariable is a Variant containing Empty , then no characters are copied. If destvariable is not convertible to a String , then a runtime error occurs. A runtime error results if destvariable is Null .
Exam- ple	<p>This example replaces a 40-character string of asterisks (*) with an RSet and LSet string and then displays the result.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim msg1,tmpstr\$ tmpstr\$ = String(40,"*") msg1 = "Here are two strings that have been right-" + crlf msg1 = msg1 & "and left-justified in a 40-character string." msg1 = msg1 & crlf & crlf RSet tmpstr\$ = "Right " msg1 = msg1 & tmpstr\$ & crlf LSet tmpstr\$ = " Left" msg1 = msg1 & tmpstr\$ & crlf MsgBox msg1 End Sub </pre>
See Also	LSet (on page 599) (statement).

RTrim, RTrim\$ (functions)

Syntax	RTrim[\$] (text)
Descrip- tion	Returns a string with the trailing spaces removed.

Comments	RTrim\$ returns a String , whereas RTrim returns a String variant. Null is returned if text is Null .
Example	<p>This example displays a left-justified string and its RTrim result.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() txt\$ = " This is text " tr\$ = RTrim(txt\$) MsgBox "Original ->" & txt\$ & "<-" & crlf & "Right Trimmed ->" & tr\$ & "<-" End Sub </pre>
See Also	LTrim , LTrim\$ (<i>on page 600</i>) (functions); Trim , Trim\$ (<i>on page 758</i>) (functions).

S

S

SaveFilename\$ (function)
SaveSetting (statement)
Screen.DlgBaseUnitsX (property)
Screen.DlgBaseUnitsY (property)
Screen.Height (property)
Screen.TwipsPerPixelX (property)
Screen.TwipsPerPixelY (property)
Screen.Width (property)
Second (function)
Seek (function)
Seek (statement)
Select...Case (statement)
SelectBox (function)
SendKeys (statement)
Set (statement)

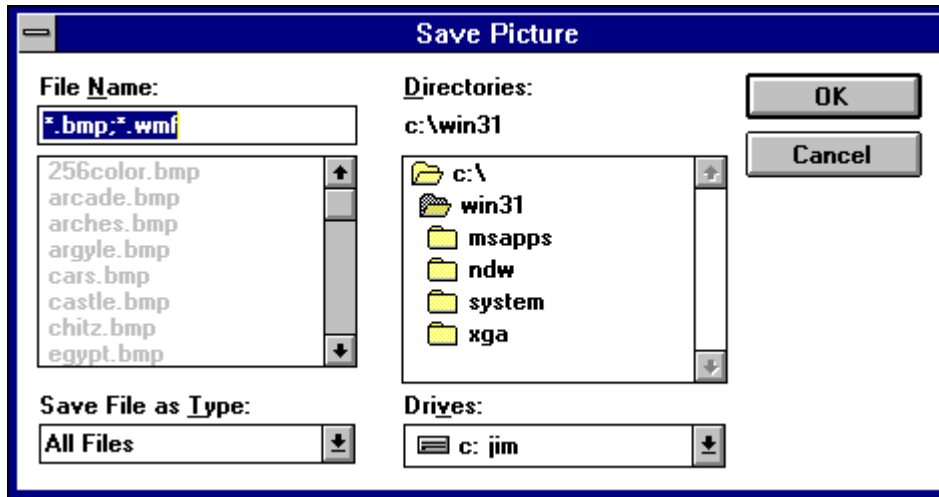
SetAttr (statement)
Sgn (function)
Shell (function)
Sin (function)
Single (data type)
Sleep (statement)
Sln (function)
Space, Space\$ (function)
Spc (function)
SQLBind (function)
SQLClose (function)
SQLError (function)
SQLExecQuery (function)
SQLGetSchema (function)
SQLOpen (function)
SQLQueryTimeout (statement)
SQLRequest (function)
SQLRetrieve (function)
SQLRetrieveToFile (function)
Sqr (function)
Stop (statement)
Str, Str\$ (functions)
StrComp (function)
StrConv (function)
String (data type)
String, String\$ (functions)
Sub...End Sub (statement)

Switch (function)
SYD (function)
System.Exit (method)
System.FreeMemory (property)
System.FreeResources (property)
System.MouseTrails (method)
System.Restart (method)
System.TotalMemory (property)
System.WindowsDirectory\$ (property)
System.WindowsVersion\$ (property)

SaveFilename\$ (function)

Syn- tax	SaveFilename\$ [(title\$ [,extensions\$])]]]	
De- scrip- tion	Displays a dialog box that prompts the user to select from a list of files and returns a String containing the full path of the selected file.	
Com- ments	The SaveFilename\$ function accepts the following parameters:	
	Parameter	Description
	title\$	String containing the title that appears on the dialog box's caption. If this string is omitted, then " Save As " is used.
	extensions\$	String containing the available file types. Its format depends on the platform on which the Basic Control Engine is running. If this string is omitted, then all files are used.
	The SaveFilename\$ function returns a full pathname of the file that the user selects. A zero-length string is returned if the user selects Cancel. If the file already exists, then the user is prompted to overwrite it.	

```
e$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
f$ = SaveFilename$("Save Picture",e$)
```



Example This example creates a save dialog box, giving the user the ability to save to several different file types.

```
Sub Main()
    e$ = "All Files:*.BMP,*.WMF;Bitmaps:*.BMP;Metafiles:*.WMF"
    f$ = SaveFilename$("Save Picture",e$)
    If Not f$ = "" Then
        MsgBox "User choose to save file as: " + f$
    Else
        MsgBox "User canceled."
    End IF
End Sub
```

See Also [MsgBox \(on page 617\)](#) (statement); [AskBox \(on page 614\)](#) (function); [AskPassword\\$ \(on page 335\)](#) (function); [InputBox, InputBox\\$ \(on page 562\)](#) (functions); [OpenFilename\\$ \(on page 646\)](#) (function); [SelectBox \(on page 709\)](#) (function); [AnswerBox \(on page 310\)](#) (function).

Note The extensions\$ parameter must be in the following format:

```
description:ext[,ext][;description:ext[,ext]]...
```

Placeholder	Description
description	Specifies the grouping of files for the user, such as All Files .

	ext	Specifies a valid file extension, such as *.BAT or *.?F? .
<p>For example, the following are valid extensions\$ specifications:</p> <pre>"All Files:*" "Documents:*.TXT,*.DOC" "All Files:*;Documents:*.TXT,*.DOC"</pre>		

SaveSetting (statement)

Syn- tax	SaveSetting appname, section, key, setting	
De- scrip- tion	Saves the value of the specified key in the system registry. The following table describes the named parameters to the SaveSetting statement:	
	Parame- ter	Description
	appname	String expression indicating the name of the application whose setting will be modified.
	section	String expression indicating the name of the section whose setting will be modified.
	key	String expression indicating the name of the setting to be modified.
	setting	The value assigned to key.
Ex- am- ple	<pre>'The following example adds two entries to the Windows registry 'if run under Win32 or to NEWAPP.INI on other platforms, 'using the SaveSetting statement. It then uses DeleteSetting 'to remove these entries. Sub Main() SaveSetting appname := "NewApp", section := "Startup", _ key := "Height", setting := 200 SaveSetting appname := "NewApp", section := "Startup", _ key := "Width", setting := 320 DeleteSetting "NewApp" 'Remove NewApp key from registry</pre>	

	<code>End Sub</code>
See Also	GetAllSettings (on page 537) (function), DeleteSetting (on page 423) (statement), GetSetting (on page 541) (function)
Note	Under Win32, this statement operates on the system registry. All settings are saved to the following entry in the system registry: <code>HKEY_CURRENT_USER\Software\BasicScript Program Settings\app-name\section\key</code> On this platform, the appname parameter is not optional.

Screen.DlgBaseUnitsX (property)

Syntax	Screen.DlgBaseUnitsX
Description	Returns an Integer used to convert horizontal pixels to and from dialog units.
Comments	The number returned depends on the name and size of the font used to display dialog boxes. To convert from pixels to dialog units in the horizontal direction: $((X\text{Pixels} * 4) + (\text{Screen.DlgBaseUnitsX} - 1)) / \text{Screen.DlgBaseUnitsX}$ To convert from dialog units to pixels in the horizontal direction: $(X\text{DlgUnits} * \text{Screen.DlgBaseUnitsX}) / 4$
Example	This example converts the screen width from pixels to dialog units. <pre>Sub Main() XPixels = Screen.Width conv% = Screen.DlgBaseUnitsX XDlgUnits = (XPixels * 4) + (conv% - 1) / conv% MsgBox "The screen width is " & XDlgUnits & " dialog units." End Sub</pre>
See Also	Screen.DlgBaseUnitsY (on page 702) (property).

Screen.DlgBaseUnitsY (property)

Syntax	Screen.DlgBaseUnitsY
--------	-----------------------------

De- scrip- tion	Returns an Integer used to convert vertical pixels to and from dialog units.
Com- ments	The number returned depends on the name and size of the font used to display dialog boxes. To convert from pixels to dialog units in the vertical direction: (YPixels * 8) + (Screen.DlgBaseUnitsY - 1) / Screen.DlgBaseUnitsY To convert from dialog units to pixels in the vertical direction: (YDlgUnits * Screen.DlgBaseUnitsY) / 8
Exam- ple	This example converts the screen width from pixels to dialog units. <pre>Sub Main() YPixels = Screen.Height conv% = Screen.DlgBaseUnitsY YDlgUnits = (YPixels * 8) + (conv% - 1) / conv% MsgBox "The screen width is " & YDlgUnits & " dialog units." End Sub</pre>
See Also	Screen.DlgBaseUnitsX (on page 702) (property).

Screen.Height (property)

Syntax	Screen.Height
De- scrip- tion	Returns the height of the screen in pixels as an Integer .
Com- ments	This property is used to retrieve the height of the screen in pixels. This value will differ depending on the display resolution. This property is read-only.
Exam- ple	This example displays the screen height in pixels. <pre>Sub Main() MsgBox "The Screen height is " & Screen.Height & " pixels." End Sub</pre>
See Also	Screen.Width (on page 704) (property).

Screen.TwipsPerPixelX (property)

Syntax	Screen.TwipsPerPixelX
Description	Returns an Integer representing the number of twips per pixel in the horizontal direction of the installed display driver.
Comments	This property is read-only.
Example	<p>This example displays the number of twips across the screen horizontally.</p> <pre> Sub Main() XScreenTwips = Screen.Width * Screen.TwipsPerPixelX MsgBox "Total horizontal screen twips = " & XScreenTwips End Sub </pre>
See Also	Screen.TwipsPerPixelY (on page 704) (property).

Screen.TwipsPerPixelY (property)

Syntax	Screen.TwipsPerPixelY
Description	Returns an Integer representing the number of twips per pixel in the vertical direction of the installed display driver.
Comments	This property is read-only.
Example	<p>This example displays the number of twips across the screen vertically.</p> <pre> Sub Main() YScreenTwips = Screen.Height * Screen.TwipsPerPixelY MsgBox "Total vertical screen twips = " & YScreenTwips End Sub </pre>
See Also	Screen.TwipsPerPixelX (on page 703) (property).

Screen.Width (property)

Syntax	Screen.Width
Description	Returns the width of the screen in pixels as an Integer .

Comments	This property is used to retrieve the width of the screen in pixels. This value will differ depending on the display resolution. This property is read-only.
Example	This example displays the screen width in pixels. <pre>Sub Main() MsgBox "The screen width is " & Screen.Width & " pixels." End Sub</pre>
See Also	Screen.Height (on page 703) (property).

Second (function)

Syntax	Second (time)
Description	Returns the second of the day encoded in the specified time parameter.
Comments	The value returned is an Integer between 0 and 59 inclusive. The time parameter is any expression that converts to a Date .
Example	This example fires an event every 10 seconds based on the system clock. <pre>Sub Main() trigger = 10 Do xs% = Second(Now) If (xs% Mod trigger = 0) Then Beep End 'Remove this line to trigger the loop continuously. Sleep 1000 End If DoEvents Loop End Sub</pre>
See Also	Day (on page 401) (function); M (on page 606) inute (on page 606) (function); Month (on page 610) (function); Year (on page 797) (function); Hour (on page 549) (function); Week-day (on page 779) (function); DatePart (on page 398) (function).

Seek (function)

Syn- tax	Seek (filenumber)	
De- scrip- tion	Returns the position of the file pointer in a file static to the beginning of the file.	
Com- ments	The filenumber parameter is a number that the Basic Control Engine uses to refer to the open file—the number passed to the Open statement. The value returned depends on the mode in which the file was opened:	
	File Mode	Returns
	Input	Byte position for the next read.
	Output	Byte position for the next write.
	Append	Byte position for the next write.
	Random	Number of the next record to be written or read.
	Binary	Byte position for the next read or write.
	The value returned is a Long between 1 and 2147483647, where the first byte (or first record) in the file is 1.	
Exam- ple	<p>This example opens a file for random write, then writes ten records into the file using the PUT statement. The file position is displayed using the Seek Function, and the file is closed.</p> <pre> Sub Main() Open "test.dat" For Random Access Write As #1 For x = 1 To 10 r% = x * 10 Put #1,x,r% Next x y = Seek(1) MsgBox "The current file position is: " & y Close End Sub </pre>	
See Also	Seek (on page 707) (statement); Loc (on page 594) (function).	

Seek (statement)

Syn- tax	Seek [#] filename,position	
De- scrip- tion	Sets the position of the file pointer within a given file such that the next read or write operation will occur at the specified position.	
Com- ments	The Seek statement accepts the following parameters:	
	Para- meter	Description
	filenum- ber	Integer used by the Basic Control Engine to refer to the open file—the number passed to the <code>Open</code> statement.
	posi- tion	Long that specifies the location within the file at which to position the file pointer. The value must be between 1 and 2147483647, where the first byte (or record number) in the file is 1. For files opened in either Binary , Output , Input , or Append mode, position is the byte position within the file. For Random files, position is the record number.
	A file can be extended by seeking beyond the end of the file and writing data there.	
Exam- ple	<p>This example opens a file for random write, then writes ten records into the file using the <code>PUT</code> statement. The file is then reopened for read, and the ninth record is read using the <code>Seek</code> and <code>Get</code> functions.</p> <pre> Sub Main() Open "test.dat" For Random Access Write As #1 For x = 1 To 10 rec\$ = "Record#: " & x Put #1,x,rec\$ Next x Close Open "test.dat" For Random Access Read As #1 Seek #1,9 Get #1,,rec\$ MsgBox "The ninth record = " & x Close </pre>	

	<pre>Kill "test.dat" End Sub</pre>
See Also	Seek (on page 706) (function); Loc (on page 594) (function)

Select...Case (statement)

Syntax	<pre>Select Case testexpression [Case expressionlist [statement_block]] [Case expressionlist [statement_block]] . . [Case Else [statement_block]] End Select</pre>	
Description	Used to execute a block of the Basic Control Engine statements depending on the value of a given expression.	
Comments	The Select Case statement has the following parts:	
	Part	Description
	test-expression	Any numeric or string expression.
	statement_block	Any group of the Basic Control Engine statements. If the testexpression matches any of the expressions contained in expressionlist, then this statement block will be executed.
	expressionlist	A comma separated list of expressions to be compared against testexpression using any of the following syntaxes: expression [, expression]... expression to expression is re-

sion- list	lational_operator expression The resultant type of expression in expressionlist must be the same as that of testexpression.
	<p>Multiple expression ranges can be used within a single Case clause. For example:</p> <pre>Case 1 to 10,12,15 Is > 40</pre> <p>Only the statement_block associated with the first matching expression will be executed. If no matching statement_block is found, then the statements following the Case Else will be executed. A Select...End Select expression can also be represented with the If...Then expression. The use of the Select statement, however, may be more readable.</p>
Exam- ple	<p>This example uses the Select...Case statement to output the current operating system.</p> <pre>Sub Main() OpSystem% = Basic.OS Select Case OpSystem% Case 0,2 s = "Microsoft Windows" Case 1 s = "DOS" Case 3 to 8,12 s = "UNIX" Case 10 s = "IBM OS/2" Case Else s = "Other" End Select MsgBox "This version of the Basic Control Engine is running on: " & s End Sub</pre>
See Also	<p>Choose (on page 362) (function); Switch (on page 744) (function); IIf (on page 555) (function); If...Then...Else (on page 553) (statement)</p>

SelectBox (function)

Syn- tax	SelectBox(title,prompt,ArrayOfItems)
-------------	--------------------------------------

De- scrip- tion	Displays a dialog box that allows the user to select from a list of choices and returns an Integer containing the index of the item that was selected.	
Com- ments	The <code>SelectBox</code> statement accepts the following parameters:	
	Para- meter	Description
	title	Title of the dialog box. This can be an expression convertible to a <code>String</code> . A runtime error is generated if title is <code>Null</code> .
	prompt	Text to appear immediately above the list box containing the items. This can be an expression convertible to a String . A runtime error is generated if prompt is <code>Null</code> .
	Array- Of- Items	Single-dimensional array. Each item from the array will occupy a single entry in the list box. A runtime error is generated if <code>ArrayOfItems</code> is not a single-dimensional array. <code>ArrayOfItems</code> can specify an array of any fundamental data type (structures are not allowed). <code>Null</code> and <code>Empty</code> values are treated as zero-length strings.
	The value returned is an Integer representing the index of the item in the list box that was selected, with 0 being the first item. If the user selects Cancel, -1 is returned.	
	<pre>result% = SelectBox("Picker", "Pick an application:", a\$)</pre>	
Exam- ple	<p>This example gets the current apps running, puts them in to an array and then asks the user to select one from a list.</p> <pre>Sub Main() Dim a\$() AppList a\$ result% = SelectBox("Picker", "Pick an application:", a\$) If Not result% = -1 then MsgBox "User selected: " & a\$(result%) Else MsgBox "User canceled" End If End Sub</pre>	
See Also	MsgBox (on page 617) (statement); AskBox\$ (on page 333) (function); AskPassword\$ (on page 335) (function); InputBox, InputBox\$ (on page 562) (functions); OpenFilename\$ (on	

	page 646) (function); SaveFilename\$ (on page 699) (function); AnswerBox (on page 310) (function)
Note	The <code>SelectBox</code> displays all text in its dialog box in 8-point MS Sans Serif.

SendKeys (statement)

Syn- tax	SendKeys KeyString\$ [[isWait] [time]]								
De- scrip- tion	Sends the specified keys to the active application, optionally waiting for the keys to be processed before continuing.								
Com- ments	The <code>SendKeys</code> statement accepts the following parameters:								
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>KeyString\$</td> <td>String containing the keys to be sent. The format for KeyString\$ is described below.</td> </tr> <tr> <td>isWait</td> <td>Boolean value. If TRUE, then the Basic Control Engine waits for the keys to be completely processed before continuing. If you are using <code>SendKeys</code> in a CimEdit/CimView script, you must set this flag to TRUE. If you do not, when a user tries to execute the <code>SendKeys</code> statement, the CimView screen freezes and processing will not continue. If FALSE (or not specified), then the BasicScript continues script execution before the active application receives all keys from the <code>SendKeys</code> statement.</td> </tr> <tr> <td>time</td> <td>Integer specifying the number of milliseconds devoted for the output of the entire KeyString\$ parameter. It must be within the following range: <div style="background-color: #f0f0f0; padding: 5px; text-align: center;"> $0 \leq \text{time} \leq 32767$ </div> For example, if time is 5000 (5 seconds) and the KeyString\$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.</td> </tr> </tbody> </table>	Parameter	Description	KeyString\$	String containing the keys to be sent. The format for KeyString\$ is described below.	isWait	Boolean value. If TRUE, then the Basic Control Engine waits for the keys to be completely processed before continuing. If you are using <code>SendKeys</code> in a CimEdit/CimView script, you must set this flag to TRUE. If you do not, when a user tries to execute the <code>SendKeys</code> statement, the CimView screen freezes and processing will not continue. If FALSE (or not specified), then the BasicScript continues script execution before the active application receives all keys from the <code>SendKeys</code> statement.	time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString\$ parameter. It must be within the following range: <div style="background-color: #f0f0f0; padding: 5px; text-align: center;"> $0 \leq \text{time} \leq 32767$ </div> For example, if time is 5000 (5 seconds) and the KeyString\$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.
Parameter	Description								
KeyString\$	String containing the keys to be sent. The format for KeyString\$ is described below.								
isWait	Boolean value. If TRUE, then the Basic Control Engine waits for the keys to be completely processed before continuing. If you are using <code>SendKeys</code> in a CimEdit/CimView script, you must set this flag to TRUE. If you do not, when a user tries to execute the <code>SendKeys</code> statement, the CimView screen freezes and processing will not continue. If FALSE (or not specified), then the BasicScript continues script execution before the active application receives all keys from the <code>SendKeys</code> statement.								
time	Integer specifying the number of milliseconds devoted for the output of the entire KeyString\$ parameter. It must be within the following range: <div style="background-color: #f0f0f0; padding: 5px; text-align: center;"> $0 \leq \text{time} \leq 32767$ </div> For example, if time is 5000 (5 seconds) and the KeyString\$ parameter contains ten keys, then a key will be output every 1/2 second. If unspecified (or 0), the keys will play back at full speed.								
	Specifying Keys To specify any key on the keyboard, simply use that key, such as "a" for lower-case a, or "A" for uppercase a . Sequences of keys are specified by appending them together: "abc" or "dir /w". Some keys have special meaning and are therefore specified in a special way, by enclosing them within braces. For example, to specify the percent sign, use "{%}" . the following table shows the special keys:								

Key	Special Meaning	Example
+	Shift	"+{F1}" 'Shift+F1
^	Ctrl	"^a" 'Ctrl+A
~	Short-cut for Enter	"~" 'Enter
%	Alt	"%F" 'Alt+F
[No special meaning	"{[}" 'Open bracket
}	Used to enclose special keys	"{Up}" 'Up Arrow
()	Used to specify grouping	"^(ab)" 'Ctrl+A, Ctrl+B
<p>Keys that are not displayed when you press them are also specified within braces, such as {Enter} or {Up}. A list of these keys follows:</p>		

{BkSp}	{BS}	{Break}	{CapsLock}	{Clear}
{Delete}	{Del}	{Down}	{End}	{Enter}
{Escape}	{Esc}	{Help}	{Home}	{Insert}

	{Left}	{NumLock}	{NumPad0}	{NumPad1}	{NumPad2}
	{NumPad3}	{NumPad4}	{NumPad5}	{NumPad6}	{NumPad7}
	{NumPad8}	{NumPad9}	{NumPad/}	{NumPad*}	{NumPad-}
	{NumPad+}	{NumPad.}	{PgDn}	{PgUp}	{PrtSc}
	{Right}	{Tab}	{Up}	{F1}	{Scroll Lock}
	{F2}	{F3}	{F4}	{F5}	{F6}
	{F7}	{F8}	{F9}	{F10}	{F11}
	{F12}	{F13}	{F14}	{F15}	{F16}
	Keys can be combined with Shift , Ctrl , and Alt using the reserved keys "+", "^", and "% " respectively: For Key Combination Use Shift+Enter "+{Enter}" Ctrl+C "^c" Alt+F2 "%{F2}"				
	To specify a modifier key combined with a sequence of consecutive keys, group the key sequence within parentheses, as in the following example: For Key Combination Use Shift+A, Shift+B "+(abc)" Ctrl+F1, Ctrl+F2 "^({F1}{F2})"				
	Use "~" as a shortcut for embedding Enter within a key sequence: For Key Combination Use a, b, Enter, d, e "ab~de" Enter, Enter "~~"				
	To embed quotation marks, use two quotation marks in a row: For Key Combination Use "Hello" ""Hello"" a"b"c "a""b""c"				
	Key sequences can be repeated using a repeat count within braces: For Key Combination Use Ten "a" keys "{a 10}" Two Enter keys "{Enter 2}"				
Ex- am- ple	<p>This example runs Notepad, writes to Notepad, and saves the new file using the SendKeys statement.</p> <pre> Sub Main() Dim id As Variant id = Shell ("notepad.exe") 'Run Notepad minimized AppActivate id 'Now activate Notepad AppMaximize 'Open and maximize the Notepad window SendKeys "Hello Notepad", 1 'Write text with time to avoid burst Sleep 2000 SendKeys "%fs", 1 'Save file (Simulate Alt+F,S keys) Sleep 2000 SendKeys "name.txt{ENTER}", 1 'Enter name of file to save </pre>				

AppClose
End Sub

Set (statement)

Syn- tax 1	Set object_var = object_expression
Syn- tax 2	Set object_var = New object_type
Syn- tax 3	Set object_var = Nothing
De- scrip- tion	Assigns a value to an object variable.
Com- ments	<p>Syntax 1 The first syntax assigns the result of an expression to an object variable. This statement does not duplicate the object being assigned but rather copies a reference of an existing object to an object variable. The object_expression is any expression that evaluates to an object of the same type as the object_var. With data objects, Set performs additional processing. When the Set is performed, the object is notified that a reference to it is being made and destroyed. For example, the following statement deletes a reference to object A , then adds a new reference to B .</p> <pre style="background-color: #e0e0e0; padding: 5px;">Set a = b</pre> <p>In this way, an object that is no longer being referenced can be destroyed.</p>
	<p>Syntax 2 In the second syntax, the object variable is being assigned to a new instance of an existing object type. This syntax is valid only for data objects. When an object created using the New keyword goes out of scope (that is, the Sub or Function in which the variable is declared ends), the object is destroyed.</p>
	<p>Syntax 3 The reserved keyword Nothing is used to make an object variable reference no object. At a later time, the object variable can be compared to Nothing to test whether the object variable has been instantiated:</p> <pre style="background-color: #e0e0e0; padding: 5px;">Set a = Nothing : If a Is Nothing Then Beep</pre>

Example	<p>This example creates two objects and sets their values.</p> <pre> Sub Main() Dim document As Object Dim page As Object Set document = GetObject("c:\resume.doc") Set page = Document.ActivePage MsgBox page.name End Sub </pre>
See Also	<p>= (on page 306) (statement); Let (on page 586) (statement); CreateObject (on page 368) (function); GetObject (on page 540) (function); Nothing (on page 628) (constant).</p>

SetAttr (statement)

Syntax	SetAttr filename\$,attribute		
Description	Changes the attribute filename\$ to the given attribute. A runtime error results if the file cannot be found.		
Comments	The SetAttr statement accepts the following parameters:		
	Parameter	Description	
	filename\$	String containing the name of the file.	
	attribute	Integer specifying the new attribute of the file.	
	The attribute parameter can contain any combination of the following values:		
	Constant	Value	Description
	ebNormal	0	Turns off all attributes
	ebReadOnly	1	Read-only files
	ebHidden	2	Hidden files
	ebSystem	4	System files
	ebVolume	8	Volume label
	ebArchive	32	Files that have changed since the last backup
	ebNone	64	Turns off all attributes

	The attributes can be combined using the + operator or the binary Or operator.
Example	<p>This example creates a file and sets its attributes to Read-Only and System.</p> <pre> Sub Main() Open "test.dat" For Output As #1 Close #1 MsgBox "The current file attribute is: " & GetAttr("test.dat") SetAttr "test.dat",ebReadOnly + ebSystem MsgBox "The file attribute was set to: " & GetAttr("test.dat") SetAttr "test.dat",ebNormal Kill "test.dat" End Sub </pre>
See Also	GetAttr (on page 538) (function); FileAttr (on page 512) (function).

Sgn (function)

Syn- tax	Sgn (number)
De- scrip- tion	Returns an Integer indicating whether a number is less than, greater than, or equal to 0.
Com- ments	Returns 1 if number is greater than 0. Returns 0 if number is equal to 0. Returns -1 if number is less than 0. The number parameter is a numeric expression of any type. If number is Null , then a runtime error is generated. Empty is treated as 0.
Exam- ple	<p>This example tests the product of two numbers and displays a message based on the sign of the result.</p> <pre> Sub Main() a% = -100 b% = 100 c% = a% * b% Select Case Sgn(c%) Case -1 MsgBox "The product is negative " & Sgn(c%) Case 0 MsgBox "The product is 0 " & Sgn(c%) </pre>

	<pre> Case 1 MsgBox "The product is positive " & Sgn(c%) End Select End Sub </pre>
See Also	Abs (on page 308) (function).

Shell (function)

Syntax	Shell (command\$ [,WindowStyle])	
Description	Executes another application, returning the task ID if successful.	
Comments	The <code>Shell</code> statement accepts the following parameters:	
	Parameter	Description
	command\$	String containing the name of the application and any parameters.
	Window-Style	Optional Integer specifying the state of the application window after execution. It can be any of the following values:
		1 Normal window with focus.
		2 Minimized with focus (default).
		3 Maximized with focus.
		4 Normal window without focus.
		7 Minimized without focus.
	An error is generated if unsuccessful running command\$. The Shell command runs programs asynchronously: the statement following the Shell statement will execute before the child application has exited. On some platforms, the next statement will run before the child application has finished loading. The Shell function returns a value suitable for activating the application using the AppActivate statement. It is important that this value be placed into a Variant , as its type depends on the platform.	

Example	<p>This example displays the Windows Clock, delays awhile, then closes it.</p> <pre> Sub Main() id = Shell("clock.exe",1) AppActivate "Clock" Sleep(2000) AppClose "Clock" End Sub </pre>
See Also	SendKeys (on page 711) (statement); AppActivate (on page 313) (statement)
Note	This function returns a global process ID that can be used to identify the new process.
Important	CIMPLICITY runs as a service. Programs started from the Event Manager run as part of the service. Services, by default, do not interact with the desktop. Therefore, shelling of a program such as CimView, will cause the program to run, but with no interface.

Sin (function)

Syntax	Sin (angle)
Description	Returns a Double value specifying the sine of angle.
Comments	The angle parameter is a Double specifying an angle in radians.
Example	<p>This example displays the sine of pi/4 radians (45 degrees).</p> <pre> Sub Main() c# = Sin(Pi / 4) MsgBox "The sine of 45 degrees is: " & c# End Sub </pre>
See Also	Tan (on page 751) (function); Cos (on page 385) (function); Atn (on page 337) (function).

Single (data type)

Syntax	Single
--------	---------------

De- scrip- tion	A data type used to declare variables capable of holding real numbers with up to seven digits of precision.
Com- ments	Single variables are used to hold numbers within the following ranges: Sign Range Negative -3.402823E38 <= single <= -1.401298E-45 Positive 1.401298E-45 <= single <= 3.402823E38 The type-declaration character for Single is ! .
	<p>Storage Internally, singles are stored as 4-byte (32-bit) IEEE values. Thus, when appearing within a structure, singles require 4 bytes of storage. When used with binary or random files, 4 bytes of storage is required. Each single consists of the following</p> <ul style="list-style-type: none"> • A 1-bit sign • An 8-bit exponent • A 24-bit mantissa
See Also	Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Integer (on page 566) (data type); Long (on page 598) (data type); Object (on page 633) (data type); String (on page 742) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement); CSng (on page 385) (function).

Sleep (statement)

Syntax	Sleep milliseconds
Description	Causes the script to pause for a specified number of milliseconds.
Comments	The milliseconds parameter is a Long in the following range: <pre>0 <= milliseconds <= 2,147,483,647</pre>
Example	This example displays a message for 2 seconds. <pre>Sub Main() MsgOpen "Waiting 2 seconds",0,False,False Sleep 2000 MsgClose End Sub</pre>

Sln (function)

Syntax	Sln (Cost,Salvage,Life)								
Description	Returns the straight-line depreciation of an asset assuming constant benefit from the asset.								
Comments	<p>The Sln of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers. The formula used to find the Sln of an asset is as follows:</p> <pre>(Cost - Salvage Value) / Useful Life</pre> <p>The Sln function requires the following parameters:</p>								
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Cost</td> <td>Double representing the initial cost of the asset.</td> </tr> <tr> <td>Salvage</td> <td>Double representing the estimated value of the asset at the end of its useful life.</td> </tr> <tr> <td>Life</td> <td>Double representing the length of the asset's useful life.</td> </tr> </tbody> </table>	Parameter	Description	Cost	Double representing the initial cost of the asset.	Salvage	Double representing the estimated value of the asset at the end of its useful life.	Life	Double representing the length of the asset's useful life.
Parameter	Description								
Cost	Double representing the initial cost of the asset.								
Salvage	Double representing the estimated value of the asset at the end of its useful life.								
Life	Double representing the length of the asset's useful life.								
	The unit of time used to express the useful life of the asset is the same as the unit of time used to express the period for which the depreciation is returned.								
Example	<p>This example calculates the straight-line depreciation of an asset that cost \$10,000.00 and has a salvage value of \$500.00 as scrap after 10 years of service life.</p> <pre>Sub Main() dep# = Sln(10000.00,500.00,10) MsgBox "The annual depreciation is: " & Format(dep#,"Currency") End Sub</pre>								
See Also	SYD (on page 745) (function); DDB (on page 402) (function).								

Space, Space\$ (functions)

Syntax	Space [\$] (NumSpaces)
--------	-------------------------------

Description	Returns a string containing the specified number of spaces.
Comments	Space\$ returns a String , whereas Space returns a String variant. NumSpaces is an Integer between 0 and 32767.
Example	This example returns a string of ten spaces and displays it. <pre>Sub Main() ln\$ = Space(10) MsgBox "Hello" & ln\$ & "over there." End Sub</pre>
See Also	String , String\$ (on page 743) (functions); Spc (on page 721) (function).

Spc (function)

Syntax	Spc (numspaces)
Description	Prints out the specified number of spaces. This function can only be used with the Print and Print# statements.
Comments	The numspaces parameter is an Integer specifying the number of spaces to be printed. It can be any value between 0 and 32767. If a line width has been specified (using the Width statement), then the number of spaces is adjusted as follows: <pre>numspaces = numspaces Mod width</pre>
	If the resultant number of spaces is greater than width - print_position , then the number of spaces is recalculated as follows: <pre>numspaces = numspaces - (width - print_position)</pre> <p>These calculations have the effect of never allowing the spaces to overflow the line length. Furthermore, with a large value for column and a small line width, the file pointer will never advance more than one line.</p>
Example	This example displays 20 spaces between the arrows. <pre>Sub Main() Print "I am"; Spc(20); "20 spaces apart!"</pre>

	<pre>Sleep (10000) 'Wait 10 seconds. End Sub</pre>
See Also	Tab (on page 750) (function); Print (on page 664) (statement); Print# (on page 666) (statement).

SQLBind (function)

Syntax	SQLBind (ID,array,column)	
Description	Specifies which fields are returned when results are requested using the <code>SQLRetrieve</code> or <code>SQLRetrieveToFile</code> function.	
Comments	The following table describes the parameters to the <code>SQLBind</code> function:	
	Parameter	Description
	ID	Long parameter specifying a valid connection.
	array	Any array of variants. Each call to <code>SQLBind</code> adds a new column number (an Integer) in the appropriate slot in the array. Thus, as you bind additional columns, the array parameter grows, accumulating a sorted list (in ascending order) of bound columns. If array is fixed, then it must be a one-dimensional variant array with sufficient space to hold all the bound column numbers. A runtime error is generated if array is too small. If array is dynamic, then it will be resized to exactly hold all the bound column numbers.
	column	Optional Long parameter that specifies the column to which to bind data. If this parameter is omitted, all bindings for the connection are dropped. <ul style="list-style-type: none"> • The first actual column in the table is column 1. • (If supported by the driver) row numbers can be returned by binding column 0.
	This function returns the number of bound columns on the connection. If no columns are bound, then 0 is returned. If there are no pending queries, then calling SQLBind will cause an error (queries are initiated using the SQLExecQuery function). If supported by the driver, row numbers can be returned by binding column 0. The Basic Control Engine generates a runtime error	

	that can be trapped if SQLBind fails. Additional error information can then be retrieved using the SQLError function.
Example	<p>This example binds columns to data.</p> <pre> Sub Main() Dim columns() As Variant id& = SQLOpen("dsn=SAMPLE",,3) t& = SQLExecQuery(id&,"Select * From c:\sample.dbf") i% = SQLBind(id&,columns,3) i% = SQLBind(id&,columns,1) i% = SQLBind(id&,columns,2) i% = SQLBind(id&,columns,6) For x = 0 To (i% - 1) MsgBox columns(x) Next x id& = SQLClose(id&) End Sub </pre>
See Also	SQLRetrieve (on page 733) (function); SQLRetrieveToFile (on page 735) (function).

SQLClose (function)

Syn-tax	SQLClose (connectionID)
De-scrip-tion	Closes the connection to the specified data source.
Com-ments	The unique connection ID (connectionID) is a Long value representing a valid connection as returned by SQLOpen . After SQLClose is called, any subsequent calls made with the connectionID will generate runtime errors. The SQLClose function returns 0 if successful; otherwise, it returns the passed connection ID and generates a trappable runtime error. Additional error information can then be retrieved using the SQLError function.
	The Basic Control Engine automatically closes all open SQL connections when either the script or the application terminates. You should use the SQLClose function rather than relying on

	the application to automatically close connections in order to ensure that your connections are closed at the proper time.
Example	<p>This example disconnects the data source sample.</p> <pre> Sub Main() Dim s As String Dim qry As Long id& = SQLOpen("dsn=SAMPLE",s\$,3) qry = LExecQuery(id&,"Select * From c:\sample.dbf") MsgBox "There are " & qry & " records in the result set." id& = SQLClose(id&) End Sub </pre>
See Also	SQLOpen (on page 730) (function).

SQLError (function)

Syntax	SQLError (ErrArray [, ID])				
Description	Retrieves driver-specific error information for the most recent SQL functions that failed.				
Comments	This function is called after any other SQL function fails. Error information is returned in a two-dimensional array (ErrArray). The following table describes the parameters to the SQLError function:				
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>ErrArray</td> <td>Two-dimensional <code>Variant</code> array, which can be dynamic or fixed. If the array is fixed, it must be (x,3), where x is the number of errors you want returned. If x is too small to hold all the errors, then the extra error information is discarded. If x is greater than the number of errors available, all errors are returned, and the empty array elements are set to <code>Empty</code>. If the array is dynamic, it will be resized to hold the exact number of errors.</td> </tr> </tbody> </table>	Parameter	Description	ErrArray	Two-dimensional <code>Variant</code> array, which can be dynamic or fixed. If the array is fixed, it must be (x,3), where x is the number of errors you want returned. If x is too small to hold all the errors, then the extra error information is discarded. If x is greater than the number of errors available, all errors are returned, and the empty array elements are set to <code>Empty</code> . If the array is dynamic, it will be resized to hold the exact number of errors.
Parameter	Description				
ErrArray	Two-dimensional <code>Variant</code> array, which can be dynamic or fixed. If the array is fixed, it must be (x,3), where x is the number of errors you want returned. If x is too small to hold all the errors, then the extra error information is discarded. If x is greater than the number of errors available, all errors are returned, and the empty array elements are set to <code>Empty</code> . If the array is dynamic, it will be resized to hold the exact number of errors.				

	ID	Optional Long parameter specifying a connection ID. If this parameter is omitted, error information is returned for the most recent SQL function call.
Each array entry in the ErrArray parameter describes one error. The three elements in each array entry contain the following information:		
	Element	Value
(entry,0)		The ODBC error state, indicated by a Long containing the error class and subclass.
(entry,1)		The ODBC native error code, indicated by a Long.
(entry,2)		The text error message returned by the driver. This field is String type.
<p>For example, to retrieve the ODBC text error message of the first returned error, the array is referenced as:</p> <pre>ErrArray(0,2)</pre> <p>The SQL_Error function returns the number of errors found. The Basic Control Engine generates a runtime error if SQL_Error fails. (You cannot use the SQL_Error function to gather additional error information in this case.)</p>		
Example	<p>This example forces a connection error and traps it for use with the SQL_Error function.</p> <pre>Sub Main() Dim a() As Variant On Error Goto Trap id& = SQLOpen("",,4) id& = SQLClose(id&) Exit Sub Trap: rc% = SQL_Error(a) If (rc%) Then For x = 0 To (rc% - 1) MsgBox "The SQL state returned was: " & a(x,0) </pre>	

<pre> MsgBox "The native error code returned was: " & a(x,1) MsgBox a(x,2) Next x End If End Sub </pre>

SQLExecQuery (function)

Syn- tax	SQLExecQuery (ID, query\$)	
De- scrip- tion	Executes an SQL statement query on a data source.	
Com- ments	This function is called after a connection to a data source is established using the SQLOpen function. The SQLExecQuery function may be called multiple times with the same connection ID, each time replacing all results. The following table describes the parameters to the SQLExecQuery function:	
	Parameter	Description
	ID	Long identifying a valid connected data source. This parameter is returned by the SQLOpen function.
	query\$	String specifying an SQL query statement. The SQL syntax of the string must strictly follow that of the driver.
	The return value of this function depends on the result returned by the SQL statement:	
	SQL Statement	Value
	SELECT...FROM	The value returned is the number of columns returned by the SQL statement.
	DELETE,INSERT,UP- DATE	The value returned is the number of rows affected by the SQL statement.
	The Basic Control Engine generates a runtime error if SQLExecQuery fails. Additional error information can then be retrieved using the SQLError function.	
Exam- ple	This example executes a query on the connected data source.	

	<pre> Sub Main() Dim s As String Dim qry As Long id& = SQLOpen("dsn=SAMPLE",s\$,3) qry = SQLExecQuery(id&,"Select * From c:\sample.dbf") MsgBox "There are " & qry & " columns in the result set." id& = SQLClose(id&) End Sub </pre>
See Also	SQLOpen (on page 730) (function); SQLClose (on page 723) (function); SQLRetrieve (on page 733) (function); SQLRetrieveToFile (on page 735) (function)

SQLGetSchema (function)

Syntax	SQLGetSchema (ID, action, [array] [qualifier\$])	
Description	Returns information about the data source associated with the specified connection.	
Comments	The following table describes the parameters to the SQLGetSchema function:	
	Parameter	Description
	ID	Long parameter identifying a valid connected data source. This parameter is returned by the SQLOpen function.
	action	Integer parameter specifying the results to be returned. The following table lists values for this parameter:
	Value	Meaning
	1	Returns a one-dimensional array of available data sources. The array is returned in the array parameter.

	2	Returns a one-dimensional array of databases (either directory names or database names, depending on the driver) associated with the current connection. The array is returned in the array parameter.
	3	Returns a one-dimensional array of owners (user IDs) of the database associated with the current connection. The array is returned in the array parameter.
	4	Returns a one-dimensional array of table names for a specified owner and database associated with the current connection. The array is returned in the array parameter.
	5	Returns a two-dimensional array (n by 2) containing information about a specified table. The array is configured as follows:
		(0,0) Zeroth column name (0,1) ODBC SQL data type (Integer) (1,0) First column name (1,1) ODBC SQL data type (Integer) : : (n-1,0) Nth column name (n-1,1) ODBC SQL data type (Integer)
	6	Returns a string containing the ID of the current user.
	7	Returns a string containing the name (either the directory name or the database name, depending on the driver) of the current database.
	8	Returns a string containing the name of the data source on the current connection.
	9	Returns a string containing the name of the DBMS of the data source on the current connection (for example, "FoxPro 2.5" or "Excel Files").
	10	Returns a string containing the name of the server for the data source.
	11	Returns a string containing the owner qualifier used by the data source (for example, "owner," "Authorization ID," "Schema").
	12	Returns a string containing the table qualifier used by the data source (for example, "table," "file").
	13	Returns a string containing the database qualifier used by the data source (for example, "database," "directory").
	14	Returns a string containing the procedure qualifier used by the data source (for example, "database procedure," "stored procedure," "procedure").
array		Optional Variant array parameter. This parameter is only required for action values 1, 2, 3, 4, and 5. The returned information is put into this array. If array is fixed and it is not the correct size necessary to hold the requested information, then SQLGetSchema will fail.

		If the array is larger than required, then any additional elements are erased. If array is dynamic, then it will be redimensioned to hold the exact number of elements requested.	
	qualifier	Optional String parameter required for actions 3, 4, or 5. The values are listed in the following table:	
		Action	Qualifier
		3	The qualifier parameter must be the name of the database represented by ID.
		4	The qualifier parameter specifies a database name and an owner name. The syntax for this string is: DatabaseName.OwnerName
		5	The qualifier parameter specifies the name of a table on the current connection.
	The Basic Control Engine generates a runtime error if SQLGetSchema fails. Additional error information can then be retrieved using the SQLException function. If you want to retrieve the available data sources (where action = 1) before establishing a connection, you can pass 0 as the ID parameter. This is the only action that will execute successfully without a valid connection.		
	This function calls the ODBC functions SQLGetInfo and SQLTables in order to retrieve the requested information. Some database drivers do not support these calls and will therefore cause the SQLGetSchema function to fail.		
Example	<p>This example gets all available data sources.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim dsn() As Variant numdims% = SQLGetSchema(0,1,dsn) If (numdims%) Then msg1 = "Valid ODBC data sources:" & crlf & crlf For x = 0 To numdims% - 1 msg1 = msg1 & dsn(x) & crlf Next x Else msg1 = "There are no available data sources." End If MsgBox msg1 End Sub </pre>		

See	SQLOpen (on page 730) (function)
Also	

SQLOpen (function)

Syntax	SQLOpen (login\$ [,completed\$] [,prompt])											
Description	Establishes a connection to the specified data source, returning a Long representing the unique connection ID.											
Comments	This function connects to a data source using a login string (login\$) and optionally sets the completed login string (completed\$) that was used by the driver. The following table describes the parameters to the SQLOpen function:											
	Parameter	Description										
	login\$	String expression containing information required by the driver to connect to the requested data source. The syntax must strictly follow the driver's SQL syntax.										
	completed\$	Optional String variable that will receive a completed connection string returned by the driver. If this parameter is missing, then no connection string will be returned.										
	prompt	Integer expression specifying any of the following values:										
		<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>The driver's login dialog box is always displayed.</td> </tr> <tr> <td>2</td> <td>The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior.</td> </tr> <tr> <td>3</td> <td>The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable.</td> </tr> <tr> <td>4</td> <td>The driver's login dialog box is never displayed.</td> </tr> </tbody> </table>	Value	Meaning	1	The driver's login dialog box is always displayed.	2	The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior.	3	The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable.	4	The driver's login dialog box is never displayed.
Value	Meaning											
1	The driver's login dialog box is always displayed.											
2	The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. This is the default behavior.											
3	The driver's dialog box is only displayed if the connection string does not contain enough information to make the connection. Dialog box options that were passed as valid parameters are dimmed and unavailable.											
4	The driver's login dialog box is never displayed.											
	The SQLOpen function will never return an invalid connection ID. The following example establishes a connection using the driver's login dialog box:											

	<pre>id& = SQLOpen("",,1)</pre> <p>The Basic Control Engine returns 0 and generates a trappable runtime error if SQLOpen fails. Additional error information can then be retrieved using the SQLError function. Before you can use any SQL statements, you must set up a data source and relate an existing database to it. This is accomplished using the <code>odbcadm.exe</code> program.</p>
Example	<p>This example connects the data source called "sample," returning the completed connection string, and then displays it.</p> <pre>Sub Main() Dim s As String id& = SQLOpen("dsn=SAMPLE",s\$,3) MsgBox "The completed connection string is: " & s\$ id& = SQLClose(id&) End Sub</pre>
See Also	SQLClose (on page 723) (function)

SQLQueryTimeout (statement)

Syntax	SQLQueryTimeout time	
Description	Specifies the timeout, in seconds, for ODBC queries. If you do not set SQLQueryTimeout , the default timeout is 60 seconds (1 minute).	
Comments	The SQLQueryTimeout statement accepts the following parameter:	
	Parameter	Description
	Time	Integer specifying the timeout for ODBC queries in seconds.
Example	<p>The following example sets the timeout for ODBC queries to 120 seconds (2 minutes).</p> <pre>Sub Main() SQLQueryTimeout 120 End Sub</pre>	

SQLRequest (function)

Syn-tax	SQLRequest (connection\$,query\$,array [[output\$] [,prompt][,isColumnNames]])	
De-scrip-tion	Opens a connection, runs a query, and returns the results as an array.	
Com-ments	The SQLRequest function takes the following parameters:	
	Para-meter	Description
	con-nection	String specifying the connection information required to connect to the data source.
	query	String specifying the query to execute. The syntax of this string must strictly follow the syntax of the ODBC driver.
	array	Array of variants to be filled with the results of the query. The array parameter must be dynamic: it will be resized to hold the exact number of records and fields.
	output	Optional String to receive the completed connection string as returned by the driver.
	prompt	Optional Integer specifying the behavior of the driver's dialog box.
	is-Column-Names	Optional Boolean specifying whether the column names are returned as the first row of results. The default is False .
	The Basic Control Engine generates a runtime error if SQLRequest fails. Additional error information can then be retrieved using the SQLException function. The SQLRequest function performs one of the following actions, depending on the type of query being performed:	
	Type of Query	Action
	SELECT	The <code>SQLRequest</code> function fills array with the results of the query, returning a Long containing the number of results placed in the array. The array is filled as follows (assuming an x by y query):
		<pre>(record 1,field 1) (record 1,field 2) : (record 1,field y)</pre>

		<pre>(record 2,field 1) (record 2,field 2) : (record 2,field y) : : (record x,field 1) (record x,field 2) : (record x,field y)</pre>
	<p>INSERT, DELETE, UPDATE</p>	<p>The SQLRequest function erases array and returns a Long containing the number of affected rows.</p>
<p>Example</p>		<p>This example opens a data source, runs a select query on it, and then displays all the data found in the result set.</p> <pre>Sub Main() Dim a() As Variant l% = SQLRequest("dsn=SAMPLE;", "Select * From c:\sample.dbf", a,,3,True) For x = 0 To Ubound(a) For y = 0 To l - 1 MsgBox a(x,y) Next y Next x End Sub</pre>

SQLRetrieve (function)

<p>Syn- tax</p>	<p>SQLRetrieve(ID,array[, [maxcolumns] [, [maxrows] [, [isColumnNames] [, isFetchFirst]])</p>
<p>De- scrip- tion</p>	<p>Retrieves the results of a query.</p>

Comments	This function is called after a connection to a data source is established, a query is executed, and the desired columns are bound. The following table describes the parameters to the SQLRetrieve function:	
	Parameter	Description
	ID	Long identifying a valid connected data source with pending query results.
	array	Two-dimensional array of variants to receive the results. The array has x rows by y columns. The number of columns is determined by the number of bindings on the connection.
	max-columns	Optional Integer expression specifying the maximum number of columns to be returned. If maxcolumns is greater than the number of columns bound, the additional columns are set to empty. If maxcolumns is less than the number of bound results, the rightmost result columns are discarded until the result fits.
	maxrows	Optional Integer specifying the maximum number of rows to be returned. If maxrows is greater than the number of rows available, all results are returned, and additional rows are set to empty. If maxrows is less than the number of rows available, the array is filled, and additional results are placed in memory for subsequent calls to SQLRetrieve .
	is-Column-Names	Optional Boolean specifying whether column names should be returned as the first row of results. The default is FALSE.
	isFetch-First	Optional Boolean expression specifying whether results are retrieved from the beginning of the result set. The default is False .
	<p>Before you can retrieve the results from a query, you must (1) initiate a query by calling the SQLExecQuery function and (2) specify the fields to retrieve by calling the SQLBind function. This function returns a Long specifying the number of rows available in the array. The Basic Control Engine generates a runtime error if SQLRetrieve fails. Additional error information is placed in memory.</p>	
Example	<p>This example executes a query on the connected data source, binds columns, and retrieves them.</p> <pre> Sub Main() Dim b() As Variant Dim c() As Variant </pre>	

<pre> id& = SQLOpen("DSN=SAMPLE",,3) qry& = SQLExecQuery(id&,"Select * From c:\sample.dbf") i% = SQLBind(id&,b,3) i% = SQLBind(id&,b,1) i% = SQLBind(id&,b,2) i% = SQLBind(id&,b,6) l& = SQLRetrieve(id&,c) For x = 0 To Ubound(c) For y = 0 To Ubound(b) MsgBox c(x,y) Next y Next x id& = SQLClose(id&) End Sub </pre>
<p>See SQLOpen (on page 730) (function); SQLExecQuery (on page 726) (function); SQLClose (on page 723) (function); SQLBind (on page 722) (function); SQLRetrieveToFile (on page 735) (function).</p>

SQLRetrieveToFile (function)

Syn-tax	SQLRetrieveToFile (ID,destination\$ [,isColumnNames] [,delimiter\$])	
De-scrip-tion	Retrieves the results of a query and writes them to the specified file.	
Com-ments	The following table describes the parameters to the SQLRetrieveToFile function:	
	Parame-ter	Description
	ID	Long specifying a valid connection ID.
	destina-tion	String specifying the file where the results are written.

	is-Column-Names	Optional Boolean specifying whether the first row of results returned are the bound column names. By default, the column names are not returned.
	delimiter	Optional String specifying the column separator. A tab (Chr\$(9)) is used as the default.
	<p>Before you can retrieve the results from a query, you must (1) initiate a query by calling the SQLExecQuery function and (2) specify the fields to retrieve by calling the SQLBind function. This function returns the number of rows written to the file. A runtime error is generated if there are no pending results or if the Basic Control Engine is unable to open the specified file. The Basic Control Engine generates a runtime error if SQLRetrieveToFile fails. Additional error information may be placed in memory for later use with the SQLError function.</p>	
Example	<p>This example opens a connection, runs a query, binds columns, and writes the results to a file.</p> <pre data-bbox="305 814 1425 1318"> Sub Main() Dim b() As Variant id& = SQLOpen("DSN=SAMPLE;UID=RICH",,4) t& = SQLExecQuery(id&,"Select * From c:\sample.dbf") i% = SQLBind(id&,b,3) i% = SQLBind(id&,b,1) i% = SQLBind(id&,b,2) i% = SQLBind(id&,b,6) l& = SQLRetrieveToFile(id&,"c:\results.txt",True,"") id& = SQLClose(id&) End Sub </pre>	
See Also	SQLOpen (on page 730) (function); SQLExecQuery (on page 726) (function); SQLClose (on page 723) (function); SQLBind (on page 722) (function); SQLRetrieve (on page 733) (function).	

Sqr (function)

Syntax	Sqr (number)
Description	Returns a Double representing the square root of number.
Comments	The number parameter is a Double greater than or equal to 0.
Example	This example calculates the square root of the numbers from 1 to 10 and displays them.

```

Const crlf = Chr$(13) + Chr$(10)

Sub Main()
    msg1 = ""
    For x = 1 To 10
        sx# = Sqr(x)
        msg1 = msg1 & "The square root of " & x & " is " &_
            Format(sx#,"Fixed") & crlf
    Next x
    MsgBox msg1
End Sub

```

Stop (statement)

Syn- tax	Stop
De- scrip- tion	Suspends execution of the current script, returning control to a debugger if one is present. If a debugger is not present, this command will have the same effect as End .
Ex- am- ple	<p>The Stop statement can be used for debugging. In this example, it is used to stop execution when Z is randomly set to 0.</p> <pre> Sub Main() For x = 1 To 10 z = Random(0,10) If z = 0 Then Stop y = x / z Next x End Sub </pre>
See Also	Exit For (on page 506) (statement); Exit Do (on page 505) (statement); Exit Function (on page 507) (statement); Exit Sub (on page 507) (statement); End (on page 487) (statement).

Str, Str\$ (functions)

Syn- tax	Str[\$] (number)
-------------	-------------------------

De- scrip- tion	Returns a string representation of the given number.
Com- ments	The number parameter is any numeric expression or expression convertible to a number. If number is negative, then the returned string will contain a leading minus sign. If number is positive, then the returned string will contain a leading space. Singles are printed using only 7 significant digits. Doubles are printed using 15–16 significant digits. These functions recognize the decimal separator and thousands separators as specified in the Regional Settings in the Control Panel. If the regional settings are changed, these functions will recognize it and act accordingly. The CStr , Format , and Format\$ functions also determine their separators based on the regional settings.
Exam- ple	In this example, the Str\$ function is used to display the value of a numeric variable. <pre>Sub Main() x# = 100.22 MsgBox "The string value is: " + Str(x#) End Sub</pre>
See Also	Format, Format\$ (on page 525) (functions); CStr (on page 386) (function).

StrComp (function)

Syn- tax	StrComp (string1,string2 [,compare])
De- scrip- tion	Returns an Integer indicating the result of comparing the two string arguments.
Com- ments	Any of the following values are returned:
	0 string1 = string2
	1 string1 > string2
	-1 string1 < string2
	NULL string1 or string2 is Null

	The <code>StrComp</code> function accepts the following parameters:	
Parameter	Description	
<code>string1</code>	First string to be compared, which can be any expression convertible to a String .	
<code>string2</code>	Second string to be compared, which can be any expression convertible to a String .	
<code>compare</code>	Optional Integer specifying how the comparison is to be performed. It can be either of the following values:	
	0	Case-sensitive comparison
	1	Case-insensitive comparison
	If <code>compare</code> is not specified, then the current Option Compare setting is used. If no Option Compare statement has been encountered, then Binary is used (that is, string comparison is case-sensitive).	
Example	<p>This example compares two strings and displays the results. It illustrates that the function compares two strings to the length of the shorter string in determining equivalency.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim abc As Integer Dim abi As Integer Dim cdc As Integer Dim cdi As Integer a\$ = "This string is UPPERCASE and lowercase" b\$ = "This string is uppercase and lowercase" c\$ = "This string" d\$ = "This string is uppercase and lowercase characters" msg1 = "a = " & a & crlf msg1 = msg1 & "b = " & b & crlf msg1 = msg1 & "c = " & c & crlf msg1 = msg1 & "d = " & d & crlf & crlf abc = StrComp(a\$,b\$,1) msg1 = msg1 & "a and c (insensitive) : " & abc & crlf abi = StrComp(a\$,b\$,0) msg1 = msg1 & "a and c (sensitive): " & abi & crlf cdc = StrComp(c\$,d\$,1) msg1 = msg1 & "c and d (insensitive): " & cdc & crlf </pre>	

	<pre> cdi = StrComp(c\$,d\$,0) msg1 = msg1 & "c and d (sensitive) : " & cdi & crlf MsgBox msg1 End Sub </pre>
See Also	Comparison Operators (on page 376) (topic); Like (on page 587) (operator); Option Compare (on page 649) (statement).

StrConv (function)

Syntax	<code>StrConv (string, conversion)</code>		
Description	Converts a string based on a conversion parameter.		
Comments	The <code>StrConv</code> function takes the following named parameters:		
	Parameter	Description	
	string	String expression specifying the string to be converted.	
	conversion	Integer specifying the types of conversions to be performed.	
	The conversion parameter can be any combination of the following constants:		
	Constant	Value	Description
	ebUpperCase	1	Converts string to uppercase. This constant is supported on all platforms.
	ebLowerCase	2	Converts string to lowercase. This constant is supported on all platforms.
	ebProperCase	3	Capitalizes the first letter of each word and lower-cases all the letters. This constant is supported on all platforms.
	ebWide	4	Converts narrow characters to wide characters. This constant is supported on Japanese locales only.

	ebNarrow	8	Converts wide characters to narrow characters. This constant is supported on Japanese locales only.
	ebKatakana	16	Converts Hiragana characters to Katakana characters. This constant is supported on Japanese locales only.
	ebHiragana	32	Converts Katakana characters to Hiragana characters. This constant is supported on Japanese locales only.
	ebUnicode	64	Converts string from MBCS to UNICODE. (This constant can only be used on platforms supporting UNICODE.)
	ebFromUnicode	128	Converts string from UNICODE to MBCS. (This constant can only be used on platforms supporting UNICODE.)
	<p>A runtime error is generated when a conversion is requested that is not supported on the current platform. For example, the <code>ebWide</code> and <code>ebNarrow</code> constants can only be used on an MBCS platform. (You can determine platform capabilities using the <code>Basic.Capabilities</code> method.) The following groupings of constants are mutually exclusive and therefore cannot be specified at the same time:</p> <pre> ebUpperCase, ebLowerCase, ebProperCase ebWide, ebNarrow ebUnicode, ebFromUnicode </pre> <p>Many of the constants can be combined. For example, <code>ebLowerCase</code> Or <code>ebNarrow</code>. When converting to proper case (i.e., the <code>ebProperCase</code> constant), the following are seen as word delimiters: tab, linefeed, carriage-return, formfeed, vertical tab, space, null.</p>		
Example	<pre> Sub Main() a = InputBox("Type any string:") MsgBox "Upper case: " & StrConv(a,ebUpperCase) MsgBox "Lower case: " & StrConv(a,ebLowerCase) MsgBox "Proper case: " & StrConv(a,ebProperCase) If Basic.Capability(10) And Basic.OS = ebWin16 Then 'This is an MBCS locale MsgBox "Narrow: " & StrConv(a,ebNarrow) MsgBox "Wide: " & StrConv(a,ebWide) MsgBox "Katakana: " & StrConv(a,ebKatakana) MsgBox "Hiragana: " & StrConv(a,ebHiragana) End If </pre>		

	End Sub
See Also	UCase, UCase\$ (on page 766) (functions), LCase, LCase\$ (on page 582) (functions), Basic.Capability (on page 339) (method)

String (data type)

Syn-tax	String
De-scrip-tion	A data type capable of holding a number of characters.
Com-ments	Strings are used to hold sequences of characters, each character having a value between 0 and 255. Strings can be any length up to a maximum length of 32767 characters. Strings can contain embedded nulls, as shown in the following example: <code>s\$ = "Hello" + Chr\$(0) + "there"</code> 'String with embedded null
	The length of a string can be determined using the Len function. This function returns the number of characters that have been stored in the string, including unprintable characters. The type-declaration character for String is \$.
	String variables that have not yet been assigned are set to zero-length by default.
	Strings are normally declared as variable-length, meaning that the memory required for storage of the string depends on the size of its content. The following script statements declare a variable-length string and assign it a value of length 5: <pre>Dim s As String s = "Hello" 'String has length 5.</pre>
	Fixed-length strings are given a length in their declaration: <code>Dim s As String * 20</code> <code>s = "Hello"</code> 'String has length 20 (internally pads with spaces).
	When a string expression is assigned to a fixed-length string, the following rules apply:
	<ul style="list-style-type: none"> • If the string expression is less than the length of the fixed-length string, then the fixed-length string is padded with spaces up to its declared length.
	<ul style="list-style-type: none"> • If the string expression is greater than the length of the fixed-length string, then the string expression is truncated to the length of the fixed-length string.

	Fixed-length strings are useful within structures when a fixed size is required, such as when passing structures to external routines.	
	The storage for a fixed-length string depends on where the string is declared, as described in the following table:	
	Strings Declared	Are Stored
	In structures	In the same data area as that of the structure. Local structures are on the stack; public structures are stored in the public data space; and private structures are stored in the private data space. Local structures should be used sparingly as stack space is limited.
	In arrays	In the global string space along with all the other array elements.
	Local routines	On the stack. The stack is limited in size, so local fixed-length strings should be used sparingly.
See Also	Currency (on page 387) (data type); Date (on page 392) (data type); Double (on page 458) (data type); Integer (on page 566) (data type); Long (on page 598) (data type); Object (on page 633) (data type); Single (on page 718) (data type); Variant (on page 771) (data type); Boolean (on page 351) (data type); DefType (on page 421) (statement); CStr (on page 386) (function).	

String, String\$ (functions)

Syntax	String[\$] (number,[CharCode text\$])	
Description	Returns a string of length number consisting of a repetition of the specified filler character.	
Comments	String\$ returns a String , whereas String returns a String variant. These functions take the following parameters:	
	Para-	Description

	me- ter	
	num- ber	Integer specifying the number of repetitions.
	Char- Code	Integer specifying the character code to be used as the filler character. If CharCode is greater than 255 (the largest character value), then the Basic Control Engine converts it to a valid character using the following formula: CharCode Mod 256
	text\$	Any String expression, the first character of which is used as the filler character.
Exam- ple	<p>This example uses the String function to create a line of "=" signs the length of another string and then displays the character string underlined with the generated string.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = "This string will appear underlined." b\$ = String(Len(a\$), "_") MsgBox a\$ & crlf & b\$ End Sub </pre>	
See Also	Space , Space\$ (on page 720) (functions).	

Sub...End Sub (statement)

1. Must start with a letter.
2. May contain letters, digits, and the underscore character (_). Punctuation and type-declaration characters are not allowed. The exclamation point (!) can appear within the name as long as it is not the last character.
3. Must not exceed 80 characters in length.
4. The call cannot end with a comma. For instance, using the above example, the following is not valid:

```
Test 1,,
```

5. The call must contain the minimum number of parameters as required by the called subroutine. For instance, using the above example, the following are invalid:

Switch (function)

Syn-tax	Switch (condition1,expression1 [,condition2,expression2 ... [,condition7,expression7])
De-scrip-tion	Returns the expression corresponding to the first True condition.
Com-ments	The Switch function evaluates each condition and expression, returning the expression that corresponds to the first condition (starting from the left) that evaluates to True . Up to seven condition/expression pairs can be specified. A runtime error is generated if there is an odd number of parameters (that is, there is a condition without a corresponding expression). The Switch function returns Null if no condition evaluates to True .
Exam-ple	<p>The following code fragment displays the current operating platform. If the platform is unknown, then the word "Unknown" is displayed.</p> <pre> Sub Main() Dim a As Variant a = Switch(Basic.OS = 0,"Windows XP",Basic.OS = 2,"Win32",Basic.OS = 11,"OS/2") MsgBox "The current platform is: " & IIf(IsNull(a),"Unknown",a) End Sub </pre>
See Also	Choose (on page 362) (function); IIf (on page 555) (function); If...Then...Else (on page 553) (statement); Select...Case (on page 708) (statement).

SYD (function)

Syn-tax	SYD (Cost,Salvage,Life,Period)
De-scrip-tion	Returns the sum of years' digits depreciation of an asset over a specific period of time.
Com-ments	<p>The SYD of an asset is found by taking an estimate of its useful life in years, assigning values to each year, and adding up all the numbers. The formula used to find the SYD of an asset is as follows:</p> <pre> (Cost - Salvage_Value) * Remaining_Useful_Life / SYD </pre> <p>The SYD function requires the following parameters:</p>

	Parameter	Description
	Cost	Double representing the initial cost of the asset.
	Salvage	Double representing the estimated value of the asset at the end of its useful life.
	Life	Double representing the length of the asset's useful life.
	Period	Double representing the period for which the depreciation is to be calculated. It cannot exceed the life of the asset.
	To receive accurate results, the parameters Life and Period must be expressed in the same units. If Life is expressed in terms of months, for example, then Period must also be expressed in terms of months.	
Example	<p>In this example, an asset that cost \$1,000.00 is depreciated over ten years. The salvage value is \$100.00, and the sum of the years' digits depreciation is shown for each year.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() msg1 = "" For x = 1 To 10 dep# = SYD(1000,100,10,x) msg1 = msg1 & "Year: " & x & " Dep: " & Format(dep#,"Currency") & crlf Next x MsgBox msg1 End Sub </pre>	
See Also	Sln (on page 720) (function); DDB (on page 402) (function)	

System.Exit (method)

Syntax	System.Exit
Description	Exits the operating environment.
Example	<p>This example asks whether the user would like to restart Windows after exiting.</p> <pre> Sub Main message\$="Restart Windows on exit?",ebYesNo,"Exit Windows" button = MsgBox message\$ </pre>

	<pre> If button = ebYes Then System.Restart 'Yes button selected. If button = ebNo Then System.Exit 'No button selected. End Sub </pre>
See Also	System.Restart (on page 748) (method).

System.FreeMemory (property)

Syntax	System.FreeMemory
Description	Returns a Long indicating the number of bytes of free memory.
Example	<p>The following example gets the free memory and converts it to kilobytes.</p> <pre> Sub Main() FreeMem& = System.FreeMemory FreeKBytes\$ = Format(FreeMem& / 1000,"##,###") MsgBox FreeKbytes\$ & " Kbytes of free memory" End Sub </pre>
See Also	System.TotalMemory (on page 748) (property); System.FreeResources (on page 747) (property); Basic.FreeMemory (on page 341) (property).

System.FreeResources (property)

Syntax	System.FreeResources
Description	Returns an Integer representing the percentage of free system resources.
Comments	The returned value is between 0 and 100.
Example	<p>This example gets the percentage of free resources.</p> <pre> Sub Main() FreeRes% = System.FreeResources MsgBox FreeRes% & "% of memory resources available." End Sub </pre>
See Also	System.TotalMemory (on page 748) (property); System.FreeMemory (on page 747) (property); Basic.FreeMemory (on page 341) (property).

System.MouseTrails (method)

Syntax	System.MouseTrails isOn
Description	Toggles mouse trails on or off.
Comments	If isOn is True , then mouse trails are turned on; otherwise, mouse trails are turned off. A runtime error is generated if mouse trails is not supported on your system.
Example	<p>This example turns on mouse trails.</p> <pre>Sub Main System.MouseTrails 1 End Sub</pre>

System.Restart (method)

Syntax	System.Restart
Description	Restarts the operating environment.
Example	<p>This example asks whether the user would like to restart Windows after exiting.</p> <pre>Sub Main button = MsgBox ("Restart Windows on exit?", ebYesNo, _ "Exit Windows") If button = ebYes Then System.Restart 'Yes button selected. If button = ebNo Then System.Exit 'No button selected. End Sub</pre>
See Also	System.Exit (on page 746) (method).

System.TotalMemory (property)

Syntax	<code>System.TotalMemory</code>
Description	Returns a Long representing the number of bytes of available free memory in Windows.
Example	This example displays the total system memory.

	<pre> Sub Main() TotMem& = System.TotalMemory TotKBytes\$ = Format(TotMem& / 1000, "##,###") MsgBox TotKbytes\$ & " Kbytes of total system memory exist" End Sub </pre>
See Also	System.FreeMemory (on page 747) (property); System.FreeResources (on page 747) (property); Basic.FreeMemory (on page 341) (property).

System.WindowsDirectory\$ (property)

Syntax	System.WindowsDirectory\$
Description	Returns the home directory of the operating environment.
Example	<p>This example displays the Windows directory.</p> <pre> Sub Main MsgBox "Windows directory = " & System.WindowsDirectory\$ End Sub </pre>
See Also	Basic.HomeDir\$ (on page 341) (property).

System.WindowsVersion\$ (property)

Syntax	System.WindowsVersion\$
Description	Returns the version of the operating environment, such as "5."
Comments	
Example	<p>This example sets the UseWin31 variable to True if the Windows version is greater than or equal to 3.1; otherwise, it sets the UseWin31 variable to False.</p> <pre> Sub Main() If Val(System.WindowsVersion\$) >= 5 Then MsgBox "You are running a Windows version 5 or later" End If End Sub </pre>

	<pre> Else MsgBox "You are running Windows version earlier than 5" End IF End Sub </pre>
See Also	Basic.Version\$ (on page 347) (property).

T

T

Tab (function)
Tan (function)
Text (statement)
TextBox (statement)
Time, Time\$ (function)
Time, Time\$ (statements)
Timer (function)
TimeSerial (function)
TimeValue (function)
Trim, Trim\$, LTrim, LTrim\$, RTrim, RTrim\$ (functions)
True (constant)
Type (statement)
TypeName (function)
TypeOf (function)

Tab (function)

Syn- tax	Tab (column)
-------------	-----------------------

De- scrip- tion	Prints the number of spaces necessary to reach a given column position.
Com- ments	<p>This function can only be used with the Print and Print# statements. The column parameter is an Integer specifying the desired column position to which to advance. It can be any value between 0 and 32767 inclusive. Rule 1: If the current print position is less than or equal to column, then the number of spaces is calculated as:</p> <pre>column - print_position</pre>
	<p>Rule 2: If the current print position is greater than column, then column - 1 spaces are printed on the next line. If a line width is specified (using the Width statement), then the column position is adjusted as follows before applying the above two rules:</p> <pre>column = column Mod width</pre> <p>The Tab function is useful for making sure that output begins at a given column position, regardless of the length of the data already printed on that line.</p>
Exam- ple	<p>This example prints three column headers and three numbers aligned below the column headers.</p> <pre>Sub Main() Print "Column1";Tab(10);"Column2";Tab(20);"Column3" Print Tab(3);"1";Tab(14);"2";Tab(24);"3" Sleep(10000) 'Wait 10 seconds. End Sub</pre>
See Also	Spc (on page 721) (function); Print (on page 664) (statement); Print# (on page 666) (statement).

Tan (function)

Syntax	Tan (angle)
Description	Returns a Double representing the tangent of angle.
Comments	The angle parameter is a Double value given in radians.
Example	This example computes the tangent of pi/4 radians (45 degrees).

	<pre>Sub Main() c# = Tan(Pi / 4) MsgBox "The tangent of 45 degrees is: " & c# End Sub</pre>
See Also	Sin (on page 718) (function); Cos (on page 385) (function); Atn (on page 337) (function).

Text (statement)

Syn-tax	Text x,y,width,height,title\$ [[.Identifier] [[FontName\$] [[size] [,style]]]]	
De-scription	Defines a text control within a dialog box template. The text control only displays text; the user cannot set the focus to a text control or otherwise interact with it.	
Com-ments	The text within a text control word-wraps. Text controls can be used to display up to 32K of text. The Text statement accepts the following parameters:	
	Para-meter	Description
	x, y	Integer positions of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer dimensions of the control in dialog units.
	title\$	String containing the text that appears within the text control. This text may contain an ampersand character to denote an accelerator letter, such as "&Save" for Save . Pressing this accelerator letter sets the focus to the control following the Text statement in the dialog box template.
	Identifi-er	Name by which this control can be referenced by statements in a dialog function (such as <code>DlgFocus</code> and <code>DlgEnable</code>). If omitted, then the first two words from title\$ are used.
	Font-Name\$	Name of the font used for display of the text within the text control. If omitted, then the default font for the dialog is used.
	size	Size of the font used for display of the text within the text control. If omitted, then the default size for the default font of the dialog is used.

	style	Style of the font used for display of the text within the text control. This can be any of the following values:	
		ebRegular	Normal font (that is, neither bold nor italic)
		ebBold	Bold font
		ebItalic	Italic font
		ebBoldItalic	Bold-italic font
		If omitted, then ebRegular is used.	
Example	<pre> Sub Main() Begin Dialog UserDialog 81,64,128,60,"Untitled" CancelButton 80,32,40,14 OKButton 80,8,40,14 Text 4,8,68,44,"This text is displayed in the dialog box." End Dialog Dim d As UserDialog Dialog d End Sub </pre>		
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); TextBox (on page 753) (statement); Begin Dialog (on page 348) (statement), PictureButton (on page 659) (statement).		
Note	Accelerators are underlined, and the Alt+letter accelerator combination is used. 8-point MS Sans Serif is the default font used within user dialogs.		

TextBox (statement)

Syntax	TextBox x,y,width,height,.Identifier [, <u>[isMultiline]</u> [, <u>[FontName\$]</u> [, <u>[size]</u> [, <u>style</u>]]]]
Description	Defines a single or multiline text-entry field within a dialog box template.

Com- ments	If <code>isMultiline</code> is 1, the TextBox statement creates a multiline text-entry field. When the user types into a multiline field, pressing the Enter key creates a new line rather than selecting the default button. This statement can only appear within a dialog box template (that is, between the Begin Dialog and End Dialog statements). The TextBox statement requires the following parameters:	
	Parameter	Description
	x, y	Integer position of the control (in dialog units) static to the upper left corner of the dialog box.
	width, height	Integer dimensions of the control in dialog units.
	Identifier	Name by which this control can be referenced by statements in a dialog function (such as DlgFocus and DlgEnable). This parameter also creates a string variable whose value corresponds to the content of the text box. This variable can be accessed using the syntax: <pre>DialogVariable . Identifier</pre>
	isMultiline	Specifies whether the text box can contain more than a single line (0 = single-line; 1 = multiline).
	FontName\$	Name of the font used for display of the text within the text box control. If omitted, then the default font for the dialog is used.
	size	Size of the font used for display of the text within the text box control. If omitted, then the default size for the default font of the dialog is used.
	style	Style of the font used for display of the text within the text box control. This can be any of the following values:
		<code>ebRegular</code> Normal font (i.e., neither bold nor italic)
		<code>ebBold</code> Bold font
		<code>ebItalic</code> Italic font
		<code>ebBoldItalic</code> Bold-italic font
		If omitted, then <code>ebRegular</code> is used.
	When the dialog box is created, the Identifier variable is used to set the initial content of the text box. When the dialog box is dismissed, the variable will contain the new content of the text box.	

	A single-line text box can contain up to 256 characters. The length of text in a multiline text box is not limited by the Basic Control Engine; the default memory limit specified by the given platform is used instead.
Example	<pre> Sub Main() Begin Dialog UserDialog 81,64,128,60,"Untitled" CancelButton 80,32,40,14 OKButton 80,8,40,14 TextBox 4,8,68,44,.TextBox1,1 End Dialog Dim d As UserDialog d.TextBox1 = "Enter text before invoking" 'Display text in the Textbox by setting the default value of the TextBox before showing it. Dialog d End Sub </pre>
See Also	CancelButton (on page 365) (statement); CheckBox (on page 360) (statement); ComboBox (on page 373) (statement); Dialog (on page 424) (function); Dialog (on page 426) (statement); DropListBox (on page 458) (statement); GroupBox (on page 544) (statement); ListBox (on page 591) (statement); OKButton (on page 638) (statement); OptionButton (on page 652) (statement); OptionGroup (on page 653) (statement); Picture (on page 657) (statement); PushButton (on page 671) (statement); Text (on page 752) (statement); Begin Dialog (on page 348) (statement), PictureButton (on page 659) (statement).
Note	8-point MS Sans Serif is the default font used within user dialogs.

Time, Time\$ (functions)

Syntax	Time[\$]([<i>i</i>])
Description	Returns the system time as a String or as a Date variant.
Comments	The Time\$ function returns a String contains the time in 24-hour time format, whereas Time returns a Date variant. To set the time, use the Time/Time\$ statements.
Example	This example returns the system time and displays it in a dialog box.

	<pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() oldtime\$ = Time msg1 = "Time was: " & oldtime\$ & crlf Time = "10:30:54" msg1 = msg1 & "Time set to: " & Time & crlf Time = oldtime\$ msg1 = msg1 & "Time restored to: " & Time MsgBox msg1 End Sub </pre>
See	Time, Time\$ (on page 756) (statements); Date, Date\$ (on page 393) (functions); Date, Date\$ (on page 394) (statements); Now (on page 629) (function).

Time, Time\$ (statements)

Syn- tax	Time[<i>\$</i>] = newtime
De- scrip- tion	Sets the system time to the time contained in the specified string.
Com- ments	The Time\$ statement requires a string variable in one of the following formats: HH HH:MM HH:MM:SS where HH is between 0 and 23, MM is between 0 and 59, and SS is between 0 and 59. The Time statement converts any valid expression to a time, including string and numeric values. Unlike the Time\$ statement, Time recognizes many different time formats, including 12-hour times.
Exam- ple	<p>This example returns the system time and displays it in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() oldtime\$ = Time msg1 = "Time was: " & oldtime\$ & crlf Time = "10:30:54" msg1 = msg1 & "Time set to: " & Time & crlf Time = oldtime\$ msg1 = msg1 & "Time restored to: " & Time </pre>

	<pre>MsgBox msg1 End Sub</pre>
See Also	Time, Time\$ (on page 756) (statements); Date, Date\$ (on page 393) (functions); Date, Date\$ (on page 394) (statements); Now (on page 629) (function).
Note:	If you do not have permission to change the time, a runtime error 70 will be generated.

Timer (function)

Syntax	<code>Timer</code>
Description	Returns a Single representing the number of seconds that have elapsed since midnight.
Example	<p>This example displays the elapsed time between execution start and the time you clicked the OK button on the first message.</p> <pre>Sub Main() start& = Timer MsgBox "Click the OK button, please." total& = Timer - start& MsgBox "The elapsed time was: " & total& & " seconds." End Sub</pre>
See Also	Time, Time\$ (on page 755) (functions); Now (on page 629) (function).

TimeSerial (function)

Syntax	<code>TimeSerial(hour,minute,second)</code>	
Description	Returns a Date variant representing the given time with a date of zero.	
Comments	The TimeSerial function requires the following parameters:	
	Parameter	Description
	hour	Integer between 0 and 23.
	minute	Integer between 0 and 59.
	second	Integer between 0 and 59.

<p>Example</p>	<pre>Sub Main() start# = TimeSerial(10,22,30) finish# = TimeSerial(10,35,27) dif# = Abs(start# - finish#) MsgBox "The time difference is: " & Format(dif#, "hh:mm:ss") End Sub</pre>
<p>See Also</p>	<p>DateValue (on page 401) (function); TimeValue (on page 758) (function); DateSerial (on page 400) (function).</p>

TimeValue (function)

<p>Syn- tax</p>	<p>TimeValue (time_string\$)</p>
<p>De- scrip- tion</p>	<p>Returns a Date variant representing the time contained in the specified string argument.</p>
<p>Com- ments</p>	<p>This function interprets the passed time_string\$ parameter looking for a valid time specification. The time_string\$ parameter can contain valid time items separated by time separators such as colon (:) or period (.).</p>
	<p>Time strings can contain an optional date specification, but this is not used in the formation of the returned value. If a particular time item is missing, then it is set to 0. For example, the string "10 pm" would be interpreted as "22:00:00."</p>
<p>Exam- ple</p>	<p>This example calculates the TimeValue of the current time and displays it in a dialog box.</p> <pre>Sub Main() t1\$ = "10:15" t2# = TimeValue(t1\$) MsgBox "The TimeValue of " & t1\$ & " is: " & t2# End Sub</pre>
<p>See Also</p>	<p>DateValue (on page 401) (function); TimeSerial (on page 757) (function); DateSerial (on page 400) (function).</p>

Trim, Trim\$, LTrim, LTrim\$, RTrim, RTrim\$ (functions)

<p>Syntax</p>	<p>Trim[\$](string) LTrim[\$](string) RTrim[\$](string)</p>
----------------------	---

Description	Functions return the following.	
	Function	Returns
	Trim	Copy of the passed string expression (string) with both the leading and trailing spaces removed.
	LTrim	String with the leading spaces removed,
	RTrim	String with the trailing spaces removed.
	Trim\$, LTrim\$, and RTrim\$	String
	Trim, LTrim, and RTrim	String variant.
Null is returned if string is Null.		
Comments	Trim\$ returns a String , whereas Trim returns a String variant. Null is returned if text is Null.	
Example 1	<pre> This first example uses the Trim\$ function to extract the nonblank part of a string and display it. Const crlf = Chr\$(13) + Chr\$(10) Sub Main() text\$ = " This is text " tr\$ = Trim\$(text\$) MsgBox "Original =>" & text\$ & "<=" & crlf & _ "Trimmed =>" & tr\$ & "<=" End Sub </pre>	
Example 2	<pre> This second example displays a right-justified string and its LTrim result. Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = " <= This is a right-justified string" b\$ = LTrim\$(a\$) MsgBox a\$ & crlf & b\$ </pre>	

	<code>End Sub</code>
Example 3	<pre>'This third example displays a left-justified string and its 'RTrim result. Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = "This is a left-justified string. " b\$ = RTrim\$(a\$) MsgBox a\$ & "<=" & crlf & b\$ & "<=" End Sub</pre>
See Also	LTrim, LTrim\$ (on page 600) (functions); RTrim, RTrim\$ (on page 696) (functions).

True (constant)

Description	Boolean constant whose value is True .
Comments	Used in conditionals and Boolean expressions.
Example	<p>This example sets variable a to True and then tests to see whether (1) A is True; (2) the True constant = -1; and (3) A is equal to -1 (True).</p> <pre>Sub Main() a = True If ((a = True) and (True = -1) and (a = -1)) then MsgBox "a is True." Else MsgBox "a is False." End If End Sub</pre>
See Also	False (on page 511) (constant); Constants (topic); Boolean (on page 351) (data type).

Type (statement)

Syntax	<pre>Type username variable As type variable As type variable As type : End Type</pre>
Description	<p>The Type statement creates a structure definition that can then be used with the Dim statement to declare variables of that type. The username field specifies the name of the structure that is used later with the Dim statement.</p>
Comments	<p>Within a structure definition appear field descriptions in the format:</p> <pre>variable As type</pre> <p>where variable is the name of a field of the structure, and type is the data type for that variable. Any fundamental data type or previously declared user-defined data type can be used within the structure definition (structures within structures are allowed). Only fixed arrays can appear within structure definitions. The Type statement can only appear outside of subroutine and function declarations.</p>
	<p>When declaring strings within fixed-size types, it is useful to declare the strings as fixed-length. Fixed-length strings are stored within the structure itself rather than in the string space. For example, the following structure will always require 62 bytes of storage:</p> <pre>Type Person FirstName As String * 20 LastName As String * 40 Age As Integer End Type</pre> <p>Note: Fixed-length strings within structures are size-adjusted upward to an even byte boundary. Thus, a fixed-length string of length 5 will occupy 6 bytes of storage within the structure.</p>
Example	<p>This example displays the use of the Type statement to create a structure representing the parts of a circle and assign values to them.</p> <pre>Type Circ msg As String rad As Integer dia As Integer</pre>

	<pre> are As Double cir As Double End Type Sub Main() Dim circle As Circ circle.rad = 5 circle.dia = circle.rad * 2 circle.are = (circle.rad ^ 2) * Pi circle.cir = circle.dia * Pi circle.msg = "The area of this circle is: " & circle.are MsgBox circle.msg End Sub </pre>
See	Dim (on page 426) (statement); Public (on page 670) (statement); Private (on page 668)
Also	(statement).

TypeOf (function)

Syn- tax	<code>TypeOf objectvariable Is objecttype</code>
De- scrip- tion	Returns TRUE if <code>objectvariable</code> is the specified <code>type1</code> ; otherwise FALSE.
Com- ments	This function is used within the <code>If...Then</code> statement to determine if a variable is of a particular type. This function is particularly useful for determining the type of OLE automation objects.
Exam- ple	<pre> Sub Main() Dim a As Object Set a = CreateObject("Excel.Application") If TypeOf a Is "Application" Then MsgBox "We have an Application object." End If End Sub </pre>
See	TypeName (on page 763) (function)
Also	

TypeName (function)

Syntax	TypeName (varname)	
Description	Returns the type name of the specified variable.	
Comments	The returned string can be any of the following:	
	Returned String	Returned if varname is
	"String"	A String.
	object-type	A data object variable. In this case, objecttype is the name of the specific object type.
	"Integer"	An integer.
	"Long"	A long.
	"Single"	A single.
	"Double"	A double
	"Currency"	A currency value.
	"Date"	A date value.
	"Boolean"	A boolean value.
	"Error"	An error value.
	"Empty"	An uninitialized variable.
	"Null"	A variant containing no valid data.
	"Object"	An OLE automation object.
	"Unknown"	An unknown type of OLE automation object.
	"Nothing"	An uninitialized object variable.

	class	A specific type of OLE automation object. In this case, class is the name of the object as known to OLE.
	If Var-name is an	then
	array	the returned string can be any of the above strings follows by a empty parenthesis. For example, "Integer()" would be returned for an array of integers.
	expression	the expression is evaluated and a String representing the resultant data type is returned.
	OLE collection	TypeName returns the name of that object collection.
Example	<pre> 'The following example defines a subroutine that only accepts 'Integer variables. If not passed an Integer, it will inform 'the user that there was an error, displaying the actual type 'of variable that was passed. Sub Foo(a As Variant) If VarType(a) <> ebInteger Then MsgBox "Foo does not support " & TypeName(a) & " variables" End If End Sub </pre>	
See Also	TypeOf (on page 762) (function)	

U

U

UBound (function)
UCase, UCase\$ (functions)
Unlock (statement)
User Defined Types (topic)

UBound (function)

Syntax	UBound (ArrayVariable() [,dimension])
Description	Returns an Integer containing the upper bound of the specified dimension of the specified array variable.
Comments	<p>The dimension parameter is an integer that specifies the desired dimension. If not specified, then the upper bound of the first dimension is returned. The UBound function can be used to find the upper bound of a dimension of an array returned by an OLE automation method or property:</p> <pre>UBound(object.property [,dimension]) UBound(object.method [,dimension])</pre>
Example	<p>This example dimensions two arrays and displays their upper bounds.</p> <pre>Const crlf = Chr\$(13) + Chr\$(10) Sub Main() Dim a(5 To 12) Dim b(2 To 100,9 To 20) uba = UBound(a) ubb = UBound(b,2) MsgBox "The upper bound of a is: " & uba & crlf & " The upper bound of b is: " & ubb</pre>
	<p>This example uses Lbound and Ubound to dimension a dynamic array to hold a copy of an array redimmed by the FileList statement.</p> <pre>Dim fl\$() FileList fl\$,"*" count = Ubound(fl\$) If ArrayDims(a) Then Redim nl\$(Lbound(fl\$) To Ubound(fl\$)) For x = 1 To count nl\$(x) = fl\$(x) Next x MsgBox "The last element of the new array is: " & nl\$(count) End If End Sub</pre>

See	LBound (on page 581) (function); ArrayDims (on page 328) (function); Arrays (on page 329) (topic).
Also	

UCase, UCase\$ (functions)

Syntax	UCase[\$] (text)
Description	Returns the uppercase equivalent of the specified string.
Comments	UCase\$ returns a String , whereas UCase returns a String variant. Null is returned if text is Null .
Example	<p>This example uses the UCase\$ function to change a string from lowercase to uppercase.</p> <pre> Sub Main() a1\$ = "this string was lowercase, but was converted." a2\$ = UCase(a1\$) MsgBox a2\$ End Sub </pre>
See Also	LCase, LCase\$ (on page 582) (functions).

Unlock (statement)

Syntax	Unlock [#] filename [{record [start] To end}]	
Description	Unlocks a section of the specified file, allowing other processes access to that section of the file.	
Comments	The Unlock statement requires the following parameters:	
	Parameter	Description
	filename	Integer used by the Basic Control Script to refer to the open file—the number passed to the Open statement.
	record	Long specifying which record to unlock.
	start	Long specifying the first record within a range to be unlocked.

end	Long specifying the last record within a range to be unlocked.
For sequential files, the record, start, and end parameters are ignored: the entire file is unlocked. The section of the file is specified using one of the following:	
Syntax	Description
No record specification	Unlock the entire file.
record	Unlock the specified record number (for Random files) or byte (for Binary files).
to end	Unlock from the beginning of the file to the specified record (for Random files) or byte (for Binary files).
start to end	Unlock the specified range of records (for Random files) or bytes (for Binary files).
The unlock range must be the same as that used by the Lock statement.	
Example	<p>This example creates a file named test.dat and fills it with ten string variable records. These are displayed in a dialog box. The file is then reopened for read/write, and each record is locked, modified, rewritten, and unlocked. The new records are then displayed in a dialog box.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() a\$ = "This is record number: " b\$ = "0" rec\$ = "" msg1 = "" Open "test.dat" For Random Access Write Shared As #1 For x = 1 To 10 rec\$ = a\$ & x Lock #1,x Put #1,,rec\$ Unlock #1,x msg1 = msg1 & rec\$ & crlf Next x Close MsgBox "The records are: " & crlf & msg1 msg1 = "" Open "test.dat" For Random Access Read Write Shared As #1 For x = 1 to 10 </pre>

	<pre> rec\$ = Mid(rec\$,1,23) & (11 - x) Lock #1,x 'Lock it for our use. Put #1,x,rec\$ 'Nobody's changed it. Unlock #1,x msg1 = msg1 & rec\$ & crlf Next x MsgBox "The records are: " & crlf & msg1 Close Kill "test.dat" End Sub </pre>
See	Lock (on page 595) (statement); Open (on page 642) (statement).
Also	

User-Defined Types (topic)

User-defined types (UDTs) are structure definitions created using the **Type** statement. UDTs are equivalent to C language structures.

Declaring Structures The **Type** statement is used to create a structure definition. Type declarations must appear outside the body of all subroutines and functions within a script and are therefore global to an entire script. Once defined, a UDT can be used to declare variables of that type using the **Dim**, **Public**, or **Private** statement. The following example defines a rectangle structure:

```

Type Rect

  left As Integer

  top As Integer

  right As Integer

  bottom As Integer

End Type

:

Sub Main()

  Dim r As Rect

  :

  r.left = 10

End Sub

```

Any fundamental data type can be used as a structure member, including other user-defined types. Only fixed arrays can be used within structures.

Copying Structures UDTs of the same type can be assigned to each other, copying the contents. No other standard operators can be applied to UDTs.

```
Dim r1 As Rect
Dim r2 As Rect
:
r1 = r2
```

When copying structures of the same type, all strings in the source UDT are duplicated and references are placed into the target UDT. The **LSet** statement can be used to copy a UDT variable of one type to another:

```
LSet variable1 = variable2
```

LSet cannot be used with UDTs containing variable-length strings. The smaller of the two structures determines how many bytes get copied.

Passing Structures UDTs can be passed both to user-defined routines and to external routines, and they can be assigned. UDTs are always passed by reference. Since structures are always passed by reference, the **ByVal** keyword cannot be used when defining structure arguments passed to external routines (using **Declare**). The **ByVal** keyword can only be used with fundamental data types such as **Integer** and **String**. Passing structures to external routines actually passes a far pointer to the data structure.

Size of Structures The **Len** function can be used to determine the number of bytes occupied by a UDT:

```
Len(udt_variable_name)
```

Since strings are stored in the Basic Control Engine's data space, only a reference (currently, 2 bytes) is stored within a structure. Thus, the **Len** function may seem to return incorrect information for structures containing strings.

V

V

Val (function)
Variant (data type)
VarType (function)
Viewport.Clear (method)
Viewport.Close (method)

Viewport.Open (method)
VLine (statement)
VPage (statement)
VScroll (statement)

Val (function)

Syn- tax	Val (string_expression)						
De- scrip- tion	Converts a given string expression to a number.						
Com- ments	<p>The number parameter can contain any of the following:</p> <ul style="list-style-type: none"> • Leading minus sign (for nonhex or octal numbers only) • Hexadecimal number in the format &Hhexdigits • Octal number in the format &Ooctaldigits • Floating-point number, which can contain a decimal point and an optional exponent 						
	Spaces, tabs, and line feeds are ignored. If number does not contain a number, then 0 is returned.						
	The Val function continues to read characters from the string up to the first nonnumeric character. The Val function always returns a double-precision floating-point value. This value is forced to the data type of the assigned variable.						
Exam- ple	<p>This example inputs a number string from an InputBox and converts it to a number variable.</p> <pre> Sub Main() a\$ = InputBox("Enter anything containing a number", "Enter Number") b# = Val(a\$) MsgBox "The value is: " & b# End Sub </pre>						
	<p>'The following table shows valid strings and their numeric equivalents:</p> <table> <tr> <td>' "1 2 3"</td> <td>123</td> </tr> <tr> <td>' "12.3"</td> <td>12.3</td> </tr> <tr> <td>' "&HFFFF"</td> <td>-1</td> </tr> </table>	' "1 2 3"	123	' "12.3"	12.3	' "&HFFFF"	-1
' "1 2 3"	123						
' "12.3"	12.3						
' "&HFFFF"	-1						

	<pre>' "&O77" 63 ' "12.345E-02" .12345</pre>
See Also	CDBl (on page 356) (function); Str, Str\$ (on page 743) (functions).

Variant (data type)

Assigning to Variants Before a **Variant** has been assigned a value, it is considered empty. Thus, immediately after declaration, the **VarType** function will return **vbEmpty**. An uninitialized variant is **0** when used in numeric expressions and is a zero-length string when used within string expressions. A **Variant** is **Empty** only after declaration and before assigning it a value. The only way for a **Variant** to become **Empty** after having received a value is for that variant to be assigned to another **Variant** containing **Empty**, for it to be assigned explicitly to the constant **Empty**, or for it to be erased using the **Erase** statement. When a variant is assigned a value, it is also assigned that value's type. Thus, in all subsequent operations involving that variant, the variant will behave like the type of data it contains.

Operations on Variants Normally, a **Variant** behaves just like the data it contains. One exception to this rule is that, in arithmetic operations, variants are automatically promoted when an overflow occurs. Consider the following statements:

```
Dim a As Integer, b As Integer, c As Integer
Dim x As Variant, y As Variant, z As Variant

a% = 32767
b% = 1
c% = a% + b%      'This will overflow.

x = 32767
y = 1
z = x + y        'z becomes a Long because of Integer overflow.
```

In the above example, the addition involving **Integer** variables overflows because the result (32768) overflows the legal range for integers. With **Variant** variables, on the other hand, the addition operator recognizes the overflow and automatically promotes the result to a **Long**.

Adding Variants The + operator is defined as performing two functions: when passed strings, it concatenates them; when passed numbers, it adds the numbers. With variants, the rules are complicated because the types of the variants are not known until execution time. If you use +, you may unintentionally perform the wrong operation. It is recommended that you use the & operator if you intend to concatenate two **String** variants. This guarantees that string concatenation will be performed and not addition.

Variants That Contain No Data A **Variant** can be set to a special value indicating that it contains no valid data by assigning the **Variant** to **Null**:

```
Dim a As Variant
a = Null
```

The only way that a **Variant** becomes **Null** is if you assign it as shown above. The **Null** value can be useful for catching errors since its value propagates through an expression. **Variant Storage** Variants require 16 bytes of storage internally:

- A 2-byte type
- A 2-byte extended type for data objects
- Bytes of padding for alignment
- An 8-byte value

Unlike other data types, writing variants to **Binary** or **Random** files does not write 16 bytes. With variants, a 2-byte type is written, followed by the data (2 bytes for **Integer** and so on). **Disadvantages of Variants** The following list describes some disadvantages of variants:

1. Using variants is slower than using the other fundamental data types (that is, **Integer**, **Long**, **Single**, **Double**, **Date**, **Object**, String, Currency, and **Boolean**). Each operation involving a **Variant** requires examination of the variant's type.
2. Variants require more storage than other data types (16 bytes as opposed to 8 bytes for a **Double**, 2 bytes for an **Integer**, and so on).
3. Unpredictable behavior. You may write code to expect an **Integer** variant. At runtime, the variant may be automatically promoted to a **Long** variant, causing your code to break.

Passing Nonvariant Data to Routines Taking Variants Passing nonvariant data to a routine that is declared to receive a variant by reference prevents that variant from changing type within that routine. For example:

```
Sub Foo(v As Variant)
    v = 50           'OK.
    v = "Hello, world." 'Get a type-mismatch error here!
End Sub

Sub Main()
    Dim i As Integer
    Foo i           'Pass an integer by reference.
End Sub
```

In the above example, since an **Integer** is passed by reference (meaning that the caller can change the original value of the **Integer**), the caller must ensure that no attempt is made to change the variant's type. **Passing Variants to Routines Taking Nonvariants** Variant variables

cannot be passed to routines that accept nonvariant data by reference, as demonstrated in the following example:

```
Sub Foo(i As Integer)
End Sub

Sub Main()
  Dim a As Variant
  Foo a      'Compiler gives type-mismatch error here.
End Sub
```

VarType (function)

Syntax	VarType (variable)		
Description	Returns an Integer representing the type of data in variable.		
Comments	The variable parameter is the name of any Variant . The following table shows the different values that can be returned by VarType :		
	Value	Constant	Data Type
	0	ebEmpty	Uninitialized
	1	ebNull	No valid data
	2	ebInteger	Integer
	3	ebLong	Long
	4	ebSingle	Single
	5	ebDouble	Double
	6	ebCurrency	Currency
	7	ebDate	Date
	8	ebString	String
	9	ebObject	object (OLE automation object)
	10	ebError	User-defined error
	11	ebBoolean	Boolean

	12	ebVariant	Variant (not returned by this function)
	13	ebDataObject	Non-OLE automation object
Comments	When passed an object, the VarType function returns the type of the default property of that object. If the object has no default property, then either ebObject or ebDataObject is returned, depending on the type of variable.		
Example	<pre> Sub Main() Dim v As Variant v = 5& 'Set v to a Long. If VarType(v) = ebInteger Then MsgBox "v is an Integer." ElseIf VarType(v) = ebLong Then MsgBox "v is a Long." End If End Sub </pre>		
See Also	Empty (on page 486) (constant); Null (on page 632) (constant); Variant (on page 771) (data type).		

Viewport.Clear (method)

Syntax	<code>Viewport.Clear</code>
Description	Clears the open viewport window.
Comments	The method has no effect if no viewport is open.
Example	<pre> Sub Main() Viewport.Open Print "This will be displayed in the viewport window." Sleep 2000 Viewport.Clear Print "This will replace the previous text." Sleep 2000 Viewport.Close End Sub </pre>

See Also	Viewport.Close (on page 775) (method), Viewport.Open (on page 775) (method)
----------	---

Viewport.Close (method)

Syntax	<code>Viewport.Close</code>
Description	This method closes an open viewport window.
Comments	The method has no effect if no viewport is opened.
Example	<pre> Sub Main() Viewport.Open Print "This will be displayed in the viewport window." Sleep 2000 Viewport.Close End Sub </pre>
See Also	Viewport.Open (on page 775) (method)

Viewport.Open (method)

Syntax	<code>Viewport.Open [title [,XPos,YPos [,width,height]]]</code>	
Description	Opens a new viewport window or switches the focus to the existing viewport window.	
Comments	The <code>Viewport.Open</code> method accepts the following named :	
	Parameter	Description
	title	Specifies a String containing the text to appear in the viewport's caption.
	XPos, YPos	Specifies Integer coordinates given in twips indicating the initial position of the upper left corner of the viewport.
	width,height	Specifies Integer values indicating the initial width and height of the viewport.

	<p>If a viewport window is already open, then it is given the focus. Otherwise, a new viewport window is created. Combined with the <code>Print</code> statement, a viewport window is a convenient place to output debugging information. The viewport window is closed when the BasicScript host application is terminated. The following keys work within a viewport window:</p>																		
	<table border="1"> <thead> <tr> <th>Key</th> <th>Scrolls</th> </tr> </thead> <tbody> <tr> <td>Up</td> <td>Up by one line.</td> </tr> <tr> <td>Down</td> <td>Down by one line.</td> </tr> <tr> <td>Home</td> <td>To the first line in the viewport window.</td> </tr> <tr> <td>End</td> <td>To the last line in the viewport window.</td> </tr> <tr> <td>PgDn</td> <td>The viewport window down by one page.</td> </tr> <tr> <td>PgUp</td> <td>The viewport window up by one page.</td> </tr> <tr> <td>Ctrl+PgUp</td> <td>The viewport window left by one page.</td> </tr> <tr> <td>Ctrl+PgDn</td> <td>The viewport window right by one page.</td> </tr> </tbody> </table>	Key	Scrolls	Up	Up by one line.	Down	Down by one line.	Home	To the first line in the viewport window.	End	To the last line in the viewport window.	PgDn	The viewport window down by one page.	PgUp	The viewport window up by one page.	Ctrl+PgUp	The viewport window left by one page.	Ctrl+PgDn	The viewport window right by one page.
Key	Scrolls																		
Up	Up by one line.																		
Down	Down by one line.																		
Home	To the first line in the viewport window.																		
End	To the last line in the viewport window.																		
PgDn	The viewport window down by one page.																		
PgUp	The viewport window up by one page.																		
Ctrl+PgUp	The viewport window left by one page.																		
Ctrl+PgDn	The viewport window right by one page.																		
	<p>Only one viewport window can be open at any given time. Any scripts with <code>Print</code> statements will output information into the same viewport window. When printing to viewports, the end-of-line character can be any of the following: a carriage return, a line feed, or a carriage-return/line-feed pair. Embedded null characters are printed as spaces.</p>																		
Example	<pre> Sub Main() Viewport.Open "BasicScript Viewport",100,100,500,500 Print "This will be displayed in the viewport window." Sleep 2000 Viewport.Close End Sub </pre>																		
See Also	<p>Viewport.Close (on page 775) (method)</p>																		

VLine (statement)

Syntax	VLine [lines]
--------	----------------------

De- scrip- tion	Scrolls the window with the focus up or down by the specified number of lines.
Com- ments	The lines parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled down by one line.
Exam- ple	<p>This example prints a series of lines to the viewport, then scrolls back up the lines to the top using VLine.</p> <pre> Sub Main() "BasicScript Viewport",100,100,500,200 For i = 1 to 50 Print "This will be displayed on line#: " & i Next i MsgBox "We will now go back 40 lines..." VLine -40 MsgBox "...and here we are!" End Sub </pre>
See Al- so	VPage (on page 777) (statement); VScroll (on page 778) (statement).

VPage (statement)

Syntax	<code>VPage [pages]</code>
De- scrip- tion	Scrolls the window with the focus up or down by the specified number of pages.
Com- ments	The pages parameter is an Integer specifying the number of lines to scroll. If this parameter is omitted, then the window is scrolled down by one page.
Exam- ple	<p>This example scrolls the viewport window up five pages.</p> <pre> Sub Main() "BasicScript Viewport",100,100,500,200 For i = 1 to 500 Print "This will be displayed on line#: " & i Next i MsgBox "We will now go back 5 pages " </pre>

	<pre>VLine -5 MsgBox "...and here we are!" End Sub</pre>
See Also	VLine (on page 776) (statement); VScroll (on page 778) (statement).

VScroll (statement)

Syn- tax	<code>VScroll percentage</code>
De- scrip- tion	Sets the thumb mark on the vertical scroll bar attached to the current window.
Com- ments	The position is given as a percentage of the total range associated with that scroll bar. For example, if the percentage parameter is 50, then the thumb mark is positioned in the middle of the scroll bar.
Exam- ple	<p>This example prints a bunch of lines to the viewport, then scrolls back to the top using VScroll .</p> <pre>Sub Main() "BasicScript Viewport",100,100,500,200 For i = 1 to 50 Print "This will be displayed on line#: " & i Next i Message\$="We will now go to the the top " MsgBox Message\$ VScroll 0 VScroll 0 MsgBox " and here we are!" End Sub</pre>
See Also	VLine (on page 776) (statement); VPage (on page 777) (statement).

W

W

Weekday (function)
While...Wend (statement)
Width# (statement)
WinActivate (statement)
WinClose (statement)
WinFind (function)
WinList (statement)
WinMaximize (statement)
WinMinimize (statement)
WinMove (statement)
WinRestore (statement)
WinSize (statement)
Word\$ (function)
WordCount (function)
Write# (statement)
Writeln (statement)

Weekday (function)

Syntax	Weekday (date)
Description	Returns an Integer value representing the day of the week given by date. Sunday is 1, Monday is 2, and so on. The date parameter is any expression representing a valid date.
Example	<p>This example gets a date in an input box and displays the day of the week and its name for the date entered.</p> <pre>Sub Main() Dim a\$(7) a\$(1) = "Sunday"</pre>

	<pre> a\$(2) = "Monday" a\$(3) = "Tuesday" a\$(4) = "Wednesday" a\$(5) = "Thursday" a\$(6) = "Friday" a\$(7) = "Saturday" Reprompt: bd = InputBox("Please enter your birthday.", "Enter Birthday") If Not(IsDate.bd)) Then Goto Reprompt dt = DateValue.bd) dw = WeekDay(dt) Msgbox "You were born on day " & dw & ", which was a " & a\$(dw) End Sub </pre>
<p>See Also</p>	<p>Day (on page 401) (function); Minute (on page 606) (function); Second (on page 705) (function); Month (on page 610) (function); Year (on page 797) (function); Hour (on page 549) (function); DatePart (on page 398) (function).</p>

While...Wend (statement)

<p>Syntax</p>	<p>While condition [statements] Wend</p>
<p>Description</p>	<p>Repeats a statement or group of statements while a condition is True .</p>
<p>Comments</p>	<p>The condition is initially and then checked at the top of each iteration through the loop.</p>
<p>Example</p>	<p>This example executes a While loop until the random number generator returns a value of 1.</p> <pre> Sub Main() x% = 0 count% = 0 While x% <> 1 And count% < 500 x% = Rnd(1) If count% > 1000 Then Exit Sub Else </pre>

	<pre> count% = count% + 1 End If Wend MsgBox "The loop executed " & count% & " times." End Sub </pre>
See Also	Do...Loop (on page 454) (statement); For...Next (on page 524) (statement).
Note:	Due to errors in program logic, you can inadvertently create infinite loops in your code. You can break out of infinite loops using Ctrl+Break .

Width# (statement)

Syntax	Width# filenumber,newwidth						
Description	Specifies the line width for sequential files opened in either Output or Append mode.						
Comments	The Width# statement requires the following parameters:						
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>filenumber</td> <td>Integer used by the Basic Control Engine to refer to the open file—the number passed to the Open statement.</td> </tr> <tr> <td>newwidth</td> <td>Integer between 0 to 255 inclusive specifying the new width. If newwidth is 0, then no maximum line length is used.</td> </tr> </tbody> </table>	Parameter	Description	filenumber	Integer used by the Basic Control Engine to refer to the open file—the number passed to the Open statement.	newwidth	Integer between 0 to 255 inclusive specifying the new width. If newwidth is 0, then no maximum line length is used.
Parameter	Description						
filenumber	Integer used by the Basic Control Engine to refer to the open file—the number passed to the Open statement.						
newwidth	Integer between 0 to 255 inclusive specifying the new width. If newwidth is 0, then no maximum line length is used.						
	When a file is initially opened, there is no limit to line length. This command forces all subsequent output to the specified file to use the specified value as the maximum line length. The Width statement affects output in the following manner: if the column position is greater than 1 and the length of the text to be written to the file causes the column position to exceed the current line width, then the data is written on the next line. The Width statement also affects output of the Print command when used with the Tab and Spc functions.						
Example	This statement sets the maximum line width for file number 1 to 80 columns.						

```

Const crlf$ = Chr$(13) + Chr$(10)

Sub Main()

  Dim i,msg1,newline$

  Open "test.dat" For Output As #1 'Create data file.

  For i = 0 To 9

    Print #1,Chr(48 + i); 'Print 0-9 to test file all on same line.

  Next i

  Print #1,crlf 'New line.

  Width #1,5 'Change line width to 5.

  For i = 0 To 9 'Print 0-9 again. This time, five characters print before line wraps.

    Print #1,Chr(48 + i);

  Next I

  Close #1

  msg1 = "The effect of the Width statement is as shown below: " & crlf

  Open "test.dat" For Input As #1 'Read new file.

  Do While Not Eof(1)

    Input #1,newline$

    msg1 = msg1 & crlf$ & newline$

  Loop

  Close #1

  msg1 = msg1 & crlf$ & crlf$ & "Choose OK to remove the test file."

  MsgBox msg1 'Display effects of Width.

  Kill "test.dat"

End Sub

```

See	Print (on page 664) (statement); Print# (on page 666) (statement); Tab (on page 750)
Also	(function) ; Spc (on page 721) (function)

WinActivate (statement)

Syn- tax	WinActivate [window_name\$ window_object] [,timeout]
De- scrip- tion	Activates the window with the given name or object value.
Com- ments	The WinActivate statement requires the following parameters:

	Parameter	Description
	window_name\$	<p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre data-bbox="386 548 1421 604">WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".</p>
	window_object	<p>HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.</p>
	timeout	<p>Integer specifying the number of milliseconds for which to attempt activation of the specified window. If not specified (or 0), then only one attempt will be made to activate the window. This value is handy when you are not certain that the window you are attempting to activate has been created.</p>
<p>If window_name\$ and window_object are omitted, then no action is performed.</p>		
Example	<p>This example runs the clock.exe program by activating the Run File dialog box from within Program Manager.</p> <pre data-bbox="305 1325 1421 1604">Sub Main() WinActivate "Program Manager" Menu "File.Run" WinActivate "Program Manager Run" SendKeys "clock.exe{ENTER}" End Sub</pre>	
See Also	<p>AppActivate (on page 313) (statement).</p>	

WinClose (statement)

Syn-tax	WinClose [window_name\$ window_object]	
De-scrip-tion	Closes the given window.	
Com-ments	The WinClose statement requires the following parameters:	
	Para-me-ter	Description
	win-dow_-name\$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example: <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p>
	win-dow_-ob-ject	HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.
	If window_name\$ and window_object are omitted, then the window with the focus is closed. This command differs from the AppClose command in that this command operates on the current window rather than the current top-level window (or application).	
Exam-ple	This example closes Microsoft Word if its object reference is found. <pre>Sub Main() Dim WordHandle As HWND Set WordHandle = WinFind("Word") If (WordHandle Is Not Nothing) Then WinClose WordHandle End Sub</pre>	
See Also	WinFind (on page 785) (function)	

Note	Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.
------	---

WinFind (function)

Syntax	WinFind (name\$) As HWND
Description	Returns an object variable referencing the window having the given name.
Comments	The name\$ parameter is specified using the same format as that used by the WinActivate statement.
Example	<p>This example closes Microsoft Word if its object reference is found.</p> <pre> Sub Main() Dim WordHandle As HWND Set WordHandle = WinFind("Word") If (WordHandle Is Not Nothing) Then WinClose WordHandle End Sub </pre>
See Also	WinActivate (on page 782) (statement).

WinList (statement)

Syntax	WinList ArrayOfWindows()
Description	Fills the passed array with references to all the top-level windows.
Comments	The passed array must be declared as an array of HWND objects. The ArrayOfWindows parameter must specify either a zero- or one-dimensional dynamic array or a single-dimensional fixed array. If the array is dynamic, then it will be redimensioned to exactly hold the new number of elements. For fixed arrays, each array element is first erased, then the new elements are placed into the array. If there are fewer elements than will fit in the array, then the remaining elements are unused. A runtime error results if the array is too small to hold the new elements. After calling this function, use the LBound and UBound functions to determine the new size of the array.

Example	<p>This example minimizes all top-level windows.</p> <pre> Sub Main() Dim a() As HWND WinList a For i = 1 To UBound(a) WinMinimize a(i) Next i End Sub </pre>
See Also	WinFind (on page 785) (function).

WinMaximize (statement)

Syntax	WinMaximize [window_name\$ window_object]						
Description	Maximizes the given window.						
Comments	The <code>WinMaximize</code> statement requires the following parameters:						
	<table border="1"> <thead> <tr> <th style="text-align: left;">Parameter</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td style="vertical-align: top;">win- dow_ name\$</td> <td> <p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p> </td> </tr> <tr> <td style="vertical-align: top;">win- dow_ -</td> <td> <p>HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.</p> </td> </tr> </tbody> </table>	Parameter	Description	win- dow_ name\$	<p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p>	win- dow_ -	<p>HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.</p>
Parameter	Description						
win- dow_ name\$	<p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p>						
win- dow_ -	<p>HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.</p>						

	object
	If window_name\$ and window_object are omitted, then the window with the focus is maximized. This command differs from the AppMaximize command in that this command operates on the current window rather than the current top-level window.
Example	<p>This example maximizes all top-level windows.</p> <pre> Sub Main() Dim a() As HWND WinList a For i = 1 To UBound(a) WinMaximize a(i) Next i End Sub </pre>
See Also	WinMinimize (on page 787) (statement); WinRestore (on page 789) (statement).
Note	Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.

WinMinimize (statement)

Syntax	WinMinimize [window_name\$ window_object]				
Description	Minimizes the given window.				
Comments	The WinMinimize statement requires the following parameters:				
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>window_name\$</td> <td>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:</td> </tr> </tbody> </table>	Parameter	Description	window_name\$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:
Parameter	Description				
window_name\$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example:				

		<pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p>
	win- dow_ ob- ject	HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.
		If window_name\$ and window_object are omitted, then the window with the focus is minimized. This command differs from the AppMinimize command in that this command operates on the current window rather than the current top-level window.
Exam- ple		See example for WinList (statement).
See Also		WinMaximize (on page 786) (statement); WinRestore (on page 789) (statement).
Note		Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.

WinMove (statement)

Syn- tax	WinMove x,y [window_name\$ window_object]	
De- scrip- tion	Moves the given window to the given x,y position.	
Com- ments	The WinMove statement requires the following parameters:	
	Para- me- ter	Description
	x,y	Integer coordinates given in twips that specify the new location for the window.

win- dow_ name\$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example: <pre>WinActivate "Notepad Find"</pre> In this example, the top-level windows are searched for a window whose title contains the word "Notepad". If found, the windows owned by the top level window are searched for one whose title contains the string "Find".
win- dow_ ob- ject	HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.
	If window_name\$ and window_object are omitted, then the window with the focus is moved. This command differs from the <code>AppMove</code> command in that this command operates on the current window rather than the current top-level window. When moving child windows, remember that the x and y coordinates are static to the client area of the parent window.
Exam- ple	This example moves Program Manager to upper left corner of the screen. <pre>WinMove 0,0,"Program Manager"</pre>
See Also	WinSize (on page 790) (statement).
Note	Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.

WinRestore (statement)

Syn- tax	WinRestore [window_name\$ window_object]
De- scrip- tion	Restores the specified window to its restore state.
Com- ments	Restoring a minimized window restores that window to its screen position before it was minimized. Restoring a maximized window resizes the window to its size previous to maximizing. The WinRestore statement requires the following parameters:

Parameter	Description
window_name\$	<p>String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of window names can be specified by separating each window name with a vertical bar (), as in the following example:</p> <pre data-bbox="386 548 1419 604">WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p>
window_object	<p>HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.</p>
	<p>If window_name\$ and window_object are omitted, then the window with the focus is restored. This command differs from the AppRestore command in that this command operates on the current window rather than the current top-level window.</p>
Example	<p>This example minimizes all top-level windows except for Program Manager.</p> <pre data-bbox="305 1171 1419 1541">Sub Main() Dim a() As HWND WinList a For i = 0 To UBound(a) WinMinimize a(i) Next I WinRestore "Program Manager" End Sub</pre>
See Also	<p>WinMaximize (on page 786) (statement); WinMinimize (on page 787) (statement)</p>
Note	<p>Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.</p>

WinSize (statement)

Syntax	WinSize width,height [,window_name\$ window_object]	
Description	Resizes the given window to the specified width and height.	
Comments	The WinSize statement requires the following parameters:	
	Parameter	Description
	width,height	Integer coordinates given in twips that specify the new size of the window.
	window_name\$	String containing the name that appears on the desired application's title bar. Optionally, a partial name can be used, such as "Word" for "Microsoft Word." A hierarchy of windows can be specified by separating each window name with a vertical bar (), as in the following example: <pre>WinActivate "Notepad Find"</pre> <p>In this example, the top-level windows are searched for a window whose title contains the word "Notepad" . If found, the windows owned by the top level window are searched for one whose title contains the string "Find" .</p>
	window_object	HWND object specifying the exact window to activate. This can be used in place of the window_name\$ parameter to indicate a specific window to activate.
	If window_name\$ and window_object are omitted, then the window with the focus is resized. This command differs from the AppSize command in that this command operates on the current window rather than the current top-level window.	
Example	This example runs and resizes Notepad. <pre>Sub Main() Dim NotepadApp As HWND id = Shell("Notepad.exe") set NotepadApp = WinFind("Notepad") WinSize 4400,8500,NotepadApp End Sub</pre>	
See Also	WinMove (on page 788) (statement)	

Note	Under Windows, the current window can be an MDI child window, a pop-up window, or a top-level window.
------	---

Word\$ (function)

Syntax	Word\$ (text\$,first[,last])								
Description	Returns a String containing a single word or sequence of words between first and last.								
Comments	The <code>Word\$</code> function requires the following parameters:								
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>text\$</td> <td>String from which the sequence of words will be extracted.</td> </tr> <tr> <td>first</td> <td>Integer specifying the index of the first word in the sequence to return. If last is not specified, then only that word is returned.</td> </tr> <tr> <td>last</td> <td>Integer specifying the index of the last word in the sequence to return. If last is specified, then all words between first and last will be returned, including all spaces, tabs, and end-of-lines that occur between those words.</td> </tr> </tbody> </table>	Parameter	Description	text\$	String from which the sequence of words will be extracted.	first	Integer specifying the index of the first word in the sequence to return. If last is not specified, then only that word is returned.	last	Integer specifying the index of the last word in the sequence to return. If last is specified, then all words between first and last will be returned, including all spaces, tabs, and end-of-lines that occur between those words.
Parameter	Description								
text\$	String from which the sequence of words will be extracted.								
first	Integer specifying the index of the first word in the sequence to return. If last is not specified, then only that word is returned.								
last	Integer specifying the index of the last word in the sequence to return. If last is specified, then all words between first and last will be returned, including all spaces, tabs, and end-of-lines that occur between those words.								
	Words are separated by any non-alphanumeric characters such as spaces, tabs, end-of-lines, and punctuation. If first is greater than the number of words in text\$, then a zero-length string is returned. If last is greater than the number of words in text\$, then all words from first to the end of the text are returned.								
Example	<p>This example finds the name "Stuart" in a string and then extracts two words from the string.</p> <pre> Sub Main() s\$ = "My last name is Williams; Stuart is my surname." c\$ = Word\$(s\$,5,6) MsgBox "The extracted name is: " & c\$ End Sub </pre>								

See	Item\$ (on page 576) (function); ItemCount (on page 577) (function); Line\$ (on page 589)
Also	(function); LineCount (on page 590) (function); WordCount (on page 793) (function).

WordCount (function)

Syntax	WordCount (text\$)
Description	Returns an Integer representing the number of words in the specified text.
Comments	Words are separated by spaces, tabs, and end-of-lines.
Example	<p>This example counts the number of words in a particular string.</p> <pre> Sub Main() s\$ = "My last name is Williams; Stuart is my surname." i% = WordCount(s\$) MsgBox "" & s\$ & " has " & i% & " words." End Sub </pre>
See Also	Item\$ (on page 576) (function); ItemCount (on page 577) (function); Line\$ (on page 589) (function); LineCount (on page 590) (function); Word\$ (on page 792) (function).

Write# (statement)

Syntax	Write [#] filename [,expressionlist]				
Description	Writes a list of expressions to a given sequential file.				
Comments	The file referenced by filename must be opened in either Output or Append mode. The filename parameter is an Integer used by the Basic Control Engine to refer to the open file—the number passed to the Open statement. The following table summarizes how variables of different types are written:				
	<table border="1"> <thead> <tr> <th>Data Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Any numeric type</td> <td>Written as text. There is no leading space, and the period is always used as the decimal separator.</td> </tr> </tbody> </table>	Data Type	Description	Any numeric type	Written as text. There is no leading space, and the period is always used as the decimal separator.
Data Type	Description				
Any numeric type	Written as text. There is no leading space, and the period is always used as the decimal separator.				

	String	Written as text, enclosed within quotes.
	Empty	No data is written.
	Null	Written as #NULL# .
	Boolean	Written as #TRUE# or #FALSE# .
	Date	Written using the universal date format: #YYYY-MM-DD HH:MM:SS#
	User-de- fined errors	Written as #ERROR ErrorNumber # , where ErrorNumber is the value of the user-defined error. The word ERROR is not translated.
	The Write statement outputs variables separated with commas. After writing each expression in the list, Write outputs an end-of-line. The Write statement can only be used with files opened in Output or Append mode.	
Exam- ple	<p>This example opens a file for sequential write, then writes ten records into the file with the values 10...50. Then the file is closed and reopened for read, and the records are read with the Input statement. The results are displayed in a dialog box.</p> <pre> Sub Main() Open "test.dat" For Output Access Write As #1 For x = 1 To 10 r% = x * 10 Write #1,x,r% Next x Close msg1 = "" Open "test.dat" For Input Access Read As #1 For x = 1 To 10 Input #1,a%,b% msg1 = msg1 & "Record " & a% & ": " & b% & Basic.Eoln\$ Next x MsgBox msg1 Close End Sub </pre>	
See Also	Open (on page 642) (statement); Put (on page 673) (statement); Print# (on page 666) (statement).	

WriteIni (statement)

Syntax	WriteIni section\$,itemName\$,value\$[,filename\$]	
Description	Writes a new value into an .ini file.	
Comments	The WriteIni statement requires the following parameters:	
	Parameter	Description
	section\$	String specifying the section that contains the desired variables, such as "windows." Section names are specified without the enclosing brackets.
	itemName\$	String specifying which item from within the given section you want to change. If itemName\$ is a zero-length string (""), then the entire section specified by section\$ is deleted.
	value\$	String specifying the new value for the given item. If value\$ is a zero-length string (""), then the item specified by itemName\$ is deleted from the ini file.
	filename\$	String specifying the name of the ini file.
Example	<p>This example sets the txt extension to be associated with Notepad.</p> <pre>Sub Main() WriteIni "Extensions", "txt", "c:\windows\notepad.exe ^.txt", "win.ini" End Sub</pre>	
See Also	ReadIni\$ (on page 687) (function); ReadIniSection (on page 687) (statement)	
Note	If filename\$ is not specified, the win.ini file is used. If the filename\$ parameter does not include a path, then this statement looks for ini files in the Windows directory.	

X

X or (operator)

Syntax	expression1 Xor expression2		
Description	Performs a logical or binary exclusion on two expressions.		
Comments	If both expressions are either Boolean , Boolean variants, or NULL variants, then a logical exclusion is performed as follows:		
	If the first expression is	and the second expression is	then the result is
	TRUE	TRUE	FALSE
	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE
	FALSE	FALSE	FALSE
	If either expression is Null , then Null is returned. Binary Exclusion If the two expressions are Integer , then a binary exclusion is performed, returning an Integer result. All other numeric types (including Empty variants) are converted to Long , and a binary exclusion is then performed, returning a Long result. Binary exclusion forms a new value based on a bit-by-bit comparison of the binary representations of the two expressions according to the following table:		

	1	Xor	1	=	0	Example
	0	Xor	1	=	1	5 01101001
	1	Xor	0	=	1	6 10101010
	0	Xor	0	=	0	Xor 11000011

Example	<p>This example builds a logic table for the XOR function and displays it.</p> <pre> Const crlf = Chr\$(13) + Chr\$(10) Sub Main() msg1 = "Logic table for Xor:" & crlf & crlf For x = -1 To 0 For y = -1 To 0 z = x Xor y msg1 = msg1 & CBool(x) & " Xor " msg1 = msg1 & CBool(y) & " = " msg1 = msg1 & CBool(z) & crlf </pre>
---------	---

	<pre> Next y Next x MsgBox msg1 End Sub </pre>
See Also	Operator Precedence (on page 648) (topic); Or (on page 654) (operator); Eqv (on page 489) (operator); Imp (on page 557) (operator); And (on page 309) (operator).

Y

Year (function)

Syn-tax	Year (date)
Description	Returns the year of the date encoded in the specified date parameter. The value returned is between 100 and 9999 inclusive. The date parameter is any expression representing a valid date.
Example	<p>This example returns the current year in a dialog box.</p> <pre> Sub Main() tdate\$ = Date\$ tyear! = Year(DateValue(tdate\$)) MsgBox "The current year is " & tyear! End Sub </pre>
See Also	Day (on page 401) (function) Minute (on page 606) (function); Second (on page 705) (function); Month (on page 610) (function); Hour (on page 549) (function); Weekday (on page 779) (function); DatePart (on page 398) (function).

CIMPLICITY Extensions to Basic

CIMPLICITY Extensions to Basic

Click a category to view extensions.

64-bit... (on page 799)
Acquire... (on page 799)

Alarm... (on page 799)
Change... (on page 799)
CimChange... (on page 799)
CimEmAlarmEvent... (on page 800)
CimEmEvent... (on page 800)
CimEmPointEvent... (on page 800)
CimGetEMEvent... (on page 801)
CimIsMaster... (on page 801)
CimLogin/CimLogout... (on page 801)
CimProjectData... (on page 801)
CimRemoveUnusedPoints... (on page 801)
Do... (on page 801)
Get... (on page 801)
IsTerminalServices... (on page 802)
LogStatus... (on page 802)
Point... (on page 802)
PointGet... (on page 804)
PointSet... (on page 804)
String... (on page 804)

Trace... (on page 804)
--

64-bit

- [DoQINTMath \(function\) \(on page 854\)](#)
- [DoUQINTMath \(function\) \(on page 855\)](#)
- [GetCurTimeHR \(function\) \(on page 856\)](#)
- [GetTimeComponentsHR \(function\) \(on page 864\)](#)
- [Point.GetTimeStampHR \(statement\) \(on page 878\)](#)
- [Point.QuadValueAsString \(property, read\) \(on page 886\)](#)
- [Point.QuadValueAsString \(property, write\) \(on page 887\)](#)
- [Point.SetQuadIntValue \(function\) \(on page 897\)](#)
- [QINTFromString \(function\) \(on page 920\)](#)
- [SetTimecomponentsHR \(function\) \(on page 918\)](#)
- [StringFromQINT \(function\) \(on page 920\)](#)
- [StringFromUQINT \(function\) \(on page 921\)](#)
- [UQINTFromString \(function\) \(on page 923\)](#)

Acquire...

- [Acquire \(function\) \(on page 804\)](#)
- [Acquire, Release \(statements\) \(on page 805\)](#)

Alarm...

- [AlarmGenerate \(statement\) \(on page 806\)](#)
- [AlarmGenerateEx \(statement\) \(on page 808\)](#)
- [AlarmUpdate \(statement\) \(on page 813\)](#)
- [AlarmUpdateCA \(statement\) \(on page 814\)](#)
- [AlarmUpdateEx \(statement\) \(on page 816\)](#)

Change...

- [ChangePassword \(statement\) \(on page 820\)](#)

CimChange...

- [CimChangeApprovalData \(Object\) \(on page 821\)](#)

CimEMAlarmEvent...

- CimEMAlarmEvent (object) *(on page 823)*
- CimEMAlarmEvent.AlarmID (property, read) *(on page 821)*
- CimEMAlarmEvent.FinalState (property, read) *(on page 822)*
- CimEMAlarmEvent.GenTime (property, read) *(on page 822)*
- CimEMAlarmEvent.Message (property, read) *(on page 823)*
- CimEMAlarmEvent.PrevState (property, read) *(on page 823)*
- CimEMAlarmEvent.RefID (property, read) *(on page 824)*
- CimEMAlarmEvent.ReqAction (property, read) *(on page 824)*
- CimEMAlarmEvent.ResourceID (property, read) *(on page 825)*

CimEMEvent...

- CimEMEvent (object) *(on page 826)*
- CimEMEvent.ActionID (property, read) *(on page 825)*
- CimEMEvent.AlarmEvent (function) *(on page 825)*
- CimEMEvent.EventID (property, read) *(on page 826)*
- CimEMEvent.ObjectID (property, read) *(on page 826)*
- CimEMEvent.PointEvent *(on page 827)*
- CimEMEvent.TimeStamp (property, read) *(on page 827)*
- CimEMEvent.Type (property, read) *(on page 827)*

CimEMPointEvent...

- CimEMPointEvent (object) *(on page 829)*
- CimEMPointEvent.Id *(on page 828)*
- CimEmPointEvent.Quality (property, read) *(on page 829)*
- CimEmPointEvent.QualityAlarmed (property, read) *(on page 830)*
- CimEmPointEvent.QualityAlarms_Enabled (property, read) *(on page 830)*
- CimEmPointEvent.QualityDisable_Write (property, read) *(on page 830)*
- CimEmPointEvent.QualityIs_Available (property, read) *(on page 832)*
- CimEmPointEvent.QualityIs_In_Range (property, read) *(on page 831)*
- CimEmPointEvent.QualityLast_Upd_Man (property, read) *(on page 831)*
- CimEmPointEvent.QualityManual_Mode (property, read) *(on page 831)*
- CimEmPointEvent.QualityStale_Data (property, read) *(on page 832)*
- CimEMPointEvent.State (property, read) *(on page 833)*
- CimEMPointEvent.TimeStamp (property, read) *(on page 833)*

- [CimEmPointEvent.UserFlags](#) (property, read) *(on page 833)*
- [CimEMPointEvent.Value](#) (property, read) *(on page 834)*

CimGetEMEvent...

- [CimGetEMEvent](#) (function) *(on page 834)*

CimIsMaster...

- [CimIsMaster](#) (function) *(on page 835)*

CimLogin/CimLogout...

- [CimLogin](#) (statement) *(on page 835)*
- [CimLogout](#) (statement) *(on page 835)*

CimProjectData...

- [CimProjectData](#) (object) *(on page 852)*
- [CimProjectData.Attributes](#) (property, read/write) *(on page 836)*
- [CimProjectData.Entity](#) (property, read/write) *(on page 838)*
- [CimProjectData.Filters](#) (property, read/write) *(on page 836)*
- [CimProjectData.GetNext](#) (function) *(on page 837)*
- [CimProjectData.Project](#) (property, read/write) *(on page 853)*
- [CimProjectData.Reset](#) (method) *(on page 854)*

CimRemoveUnusedPoints

- [CimRemoveUnusedPoints](#) (method) *(on page 854)*

Do...

- [DoQINTMath](#) (function) *(on page 854)*
- [DoUQINTMath](#) (function) *(on page 855)*

Get...

- [GetCurTimeHR](#) (function) *(on page 856)*
- [GetKey](#) (function) *(on page 857)*
- [GetMemoryInfoSymbolSpace](#) (statement) *(on page 857)*

- [GetMemoryInfoStringSpaceHandles](#) (statement) *(on page 859)*
- [GetMemoryInfoStringSpace](#) (statement) *(on page 861)*
- [GetMemoryInfoPublicSpace](#) (statement) *(on page 862)*
- [GetSystemWindowsDirectory](#) (function) *(on page 864)*
- [GetTimeComponentsHR](#) (function) *(on page 864)*
- [GetTSSessionId](#) (function) *(on page 865)*

IsTerminalServices

- [IsTerminalServices](#) (function) *(on page 866)*

LogStatus

- [LogStatus](#) (property, read/write) *(on page 866)*

Point...

- [Point](#) (object) *(on page 881)*
- [Point](#) (subject) *(on page 900)*
- [Point.AlarmAck](#) (property, read) *(on page 867)*
- [Point.Cancel](#) (method) *(on page 867)*
- [Point.ChangeApproval](#) (property, write) *(on page 868)*
- [Point.ChangeApprovalInfo](#) (property, read) *(on page 869)*
- [Point.DataType](#) (property, read) *(on page 870)*
- [Point.DisplayFormat](#) (property, read) *(on page 871)*
- [Point.DownloadPassword](#) (property, read) *(on page 871)*
- [Point.Elements](#) (property, read) *(on page 871)*
- [Point.EnableAlarm](#) (method) *(on page 872)*
- [Point.Enabled](#) (property, read) *(on page 872)*
- [Point.EuLabel](#) (property, read) *(on page 873)*
- [Point.Get](#) (statement) *(on page 873)*
- [Point.GetArray](#) (statement) *(on page 873)*
- [Point.GetNext](#) (function) *(on page 875)*
- [Point.GetNext](#) (statement) *(on page 875)*
- [Point.GetQuadIntValue](#) (function) *(on page 876)*
- [Point.GetRawArray](#) (statement) *(on page 877)*
- [Point.GetTimeStampHR](#) (statement) *(on page 878)*
- [Point.GetValue](#) (property, read) *(on page 879)*
- [Point.HasEuConv](#) (property, read) *(on page 879)*

- [Point.Id](#) (property, read/write) *(on page 880)*
- [Point.InUserView](#) (property, read) *(on page 880)*
- [Point.Length](#) (property, read) *(on page 881)*
- [Point.OnAlarm](#) (statement) *(on page 882)*
- [Point.OnAlarmAck](#) (statement) *(on page 884)*
- [Point.OnChange](#) (statement) *(on page 884)*
- [Point.OnTimed](#) (statement) *(on page 885)*
- [Point.PointTypeId](#) (property, read) *(on page 886)*
- [Point.QuadValueAsString](#) (property, read) *(on page 886)*
- [Point.QuadValueAsString](#) (property, write) *(on page 887)*
- [Point.Quality](#) (property, read) *(on page 887)*
- [Point.QualityAlarmed](#) (property, read) *(on page 887)*
- [Point.QualityAlarms_Enabled](#) (property, read) *(on page 888)*
- [Point.QualityDisable_Write](#) (property, read) *(on page 888)*
- [Point.QualityIs_Available](#) (property, read) *(on page 889)*
- [Point.QualityIs_In_Range](#) (property, read) *(on page 889)*
- [Point.QualityLast_Upd_Man](#) (property, read) *(on page 889)*
- [Point.QualityManual_Mode](#) (property, read) *(on page 890)*
- [Point.QualityStale_Data](#) (property, read) *(on page 890)*
- [Point.RawValue](#) (property, read/write) *(on page 891)*
- [Point.ReadOnly](#) (property, read) *(on page 893)*
- [Point.Set](#) (statement) *(on page 893)*
- [Point.SetArray](#) (statement) *(on page 894)*
- [Point.SetElement](#) (statement) *(on page 895)*
- [Point.SetNoAudit](#) (statement) *(on page 896)*
- [Point.SetpointPriv](#) (property, read) *(on page 896)*
- [Point.SetQuadIntValue](#) (function) *(on page 897)*
- [Point.SetRawArray](#) (statement) *(on page 897)*
- [Point.SetValue](#) (property, write) *(on page 899)*
- [Point.State](#) (property, read) *(on page 899)*
- [Point.TimeStamp](#) (property, read) *(on page 904)*
- [Point.TimeStampHR](#) (property, read) *(on page 905)*
- [Point.UserFlags](#) (property, read) *(on page 905)*
- [Point.Value](#) (property, read/write) *(on page 906)*

PointGet...

- [PointGet \(function\) \(on page 906\)](#)
- [PointGetMultiple \(function\) \(on page 908\)](#)
- [PointGetNext \(function\) \(on page 910\)](#)

PointSet...

- [PointSet \(statement\) \(on page 913\)](#)
- [PointSetMultiple \(function\) \(on page 914\)](#)
- [PointSetMultipleEx \(function\) \(on page 916\)](#)

String...

- [QINTFromString \(function\) \(on page 920\)](#)
- [StringFromQINT \(function\) \(on page 920\)](#)
- [StringFromUQINT \(function\) \(on page 921\)](#)
- [UQINTFromString \(function\) \(on page 923\)](#)

Trace...

- [Trace \(statement\) \(on page 922\)](#)
- [TraceEnable/TraceDisable \(statement\) \(on page 922\)](#)

Acquire (function)

Syntax	<code>bool = Acquire(Region\$, TimeOut&)</code>
Description	<p>Acquire a Critical Section with a timeout. If the section is not acquired within the specified timeout, a value of False is returned.</p> <p>Critical Sections are used in multithreaded application to control reentrancy, protect access global data structures, and provide synchronization. Only one thread of an application can be within a critical section at a time. Since the Basic Control Engine is a multithreaded application, you may need to use critical sections to prevent race type conditions.</p> <p>Acquire and Release only work with the same process. In other words, two standalone executables cannot protect against each other using this mechanism.</p>

	<p>In the Basic Control Engine, when an event occurs, the script is started in parallel with any other currently executing scripts. If two scripts compete for the same resource in your factory (e.g. controlling a pump) you may need to use critical sections to control access.</p> <p>Unlike a C application, access to public and private variables is controlled automatically by BASIC. That is, if two threads are trying to set and get the value of a variable access to the variable is synchronous. In other words, the thread, which is reading the value, won't get a value, which is half-written by the other thread. However, if you are accessing more than one element of a global data structure and expect another thread to be accessing the data, then you must protect the access with a critical section.</p> <p>The Basic Control Engine automatically releases any critical sections held by the script when it terminates. While the script is running, you can use the Acquire and Release commands to control when a critical section is released. You must make a call to Release for each call you make to Acquire for a critical section.</p>	
Com-ments		
	Parameter	Description
	Region\$	String. A unique identifier of the region to be operated on.
	TimeOut&	Long. The time in milliseconds to wait.
Exam-ple	Prevent reentry into the routine if the script is already in progress. If the script can't acquire the region immediately, it will exit.	
	<pre> Sub Main() if Acquire("DATEIME",0) = FALSE then Exit Sub end if if Date\$ <> LastDate then LastDate = Date\$ PointSet "DATE",LastDate end if PointSet "TIME",Time\$ Release "DATEIME" End Sub </pre>	

Acquire, Release (statements)

Note: In the Basic Control Engine, when an event occurs, the script is started in parallel. If another event triggers the same script before the script ends, two scripts will be running in parallel. The **Acquire** and **Release** routines can be used to modify this behavior. Two options are available.

1. Serialize the processing. In this case, the second instance of the script waits until the first is complete and then begins execution. This is accomplished by placing an acquire statement at the start of the script.
2. Skip processing. In this case, the second instance of the script exits without performing any processing. The example in Acquire (FUNCTION) illustrated this.


AlarmGenerate (statement)


Syntax	<code>AlarmGenerate Project\$, AlarmId\$, ResourceId\$, Message\$, [, UserId\$ [, RefId\$ [, Master]]]</code>	
Description	To generate an alarm on a local or remote CIMPLICITY project.	
	Parameter	Description
	Project\$	String . The project to generate the alarm on. An empty string "" indicates the current project.
	AlarmId\$	String . The ID of the Alarm. Must be a valid alarm of type \$CIMBASIC.
	ResourceId\$	String . The Resource ID to generate the alarm against. Used to control routing of the alarm.
	Message\$ (on page)	String . The update alarm message to display. Note: This string is substituted into the first variable field of the Alarm's message. For a user-defined alarm message, this will be the first %s field in the message. For a point alarm message, it will be the first variable field (%VAL, %ID, etc.) in the alarm message. For this reason, it is not recommended that you use the AlarmMessage\$ field when updating point alarms.
	UserId\$	String (optional). The User ID that generated the alarm.
	RefId\$	String (optional). A Reference ID used to distinguish identical alarms.
	Master	BOOLEAN (optional). By default on a computer with Server Redundancy, alarms sent by the standby computer's Event Manager are ignored. To allow

	<p>an alarm to be generated from a script on a standby computer, set Master to True.</p>
<p>Alarm Message Length</p>	<p>Important: The following use of AlarmGenerate requires extra configuration for projects created before CIMPLICITY version 6.1.</p> <p>An alarm is triggered from a BCE script using AlarmGenerate. The alarm is a \$CIMBASIC type. The alarm message contains 80 characters. Example:</p> <pre> Sub Main() AlarmGenerate "TEST_AMV", "TEST_AMV_ALARM_SCRIPT", "\$SYSTEM", "123456789012345678 90123456789012345678901234567890123456789012345678901234567890" End Sub </pre> <p>For projects created before CIMPLICITY version 6.1:</p> <p>Problem</p> <p>If BASIC generates an alarm that is greater than 72 characters to a project that does not have the following solution: The project will log an error indicating there were too many fields. The alarm will be displayed with 72 characters.</p> <p>Solution</p> <p>Allow 80 characters in a BASIC alarm message. Idttop the alarm_field record. Edit the alarm_field.idt file. Change the field_len to 80 in the \$CIMBASIC record. For projects created in CIMPLICITY version 6.1 and later 80 characters are supported automatically.</p>
<p>Ex-ample</p>	<pre> Sub Main() ' Generate a single alarm with no reference Id. AlarmGenerate "BCEDEMO", "MY_ALARM_1", "\$SYSTEM", _ "Electrical Bus 1 Failure" ' Generate three of the same alarm for different resources. AlarmGenerate "BCEDEMO", "MY_ALARM_2", "RESOURCE_1", _ "Multiple Instance for each resource" AlarmGenerate "BCEDEMO", "MY_ALARM_2", "RESOURCE_2", _ "Multiple Instance for each resource" AlarmGenerate "BCEDEMO", "MY_ALARM_2", "RESOURCE_3", _ "Multiple Instance for each resource" ' Generate three of the same alarm for the same resource ' but use a different reference id. </pre>

	<pre> AlarmGenerate "BCEDEMO", "MY_ALARM_3", "RESOURCE_1", _ "Multiple Instances for RefId", "", "1" AlarmGenerate "BCEDEMO", "MY_ALARM_3", "RESOURCE_1", _ "Multiple Instances for RefId", "", "2" AlarmGenerate "BCEDEMO", "MY_ALARM_3", "RESOURCE_1", _ "Multiple Instances for RefId", "", "3" End Sub </pre>
<p>See Also</p>	<p>AlarmUpdate (statement) (on page 813)</p>
<p>Notes</p>	<p>The Alarm ID must have an Alarm Type of \$CIMBASIC otherwise the alarm message may not be displayed correctly.</p> <p>A unique alarm in CIMPLICITY is defined by the Alarm ID, Resource ID and Reference ID combination. Each unique alarm can be displayed as a distinct entry in the Alarm Viewer. Non-unique alarms are stacked, so that the user only sees the most recent occurrence. In general, the Resource ID is used to control the routing of alarms to users. The Reference ID is used by an application to distinguish between different instances of the same alarm.</p> <p>Guidelines for AlarmGenerateEx (statement) (on page 808) also apply to AlarmGenerate.</p>

AlarmGenerateEx (statement)

<p>Syntax</p>	<p><code>AlarmGenerateEx Project\$, AlarmId\$, ResourceId\$, Message\$, DateTime, IsUTC [, UserId\$ [, RefId\$ [, Master]]]</code></p>	
	<p>Parameter</p>	<p>Description</p>
	<p>Project\$</p>	<p>String. The project to generate the alarm on. An empty string "" indicates the current project</p>
	<p>AlarmId\$</p>	<p>String. The ID of a non-point or point Alarm that is listed in the right-pane of the Workbench>Alarms section.</p> <div data-bbox="532 1598 1419 1686" style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; margin: 10px 0;"> <p> Note:</p> </div> <p>Non-point alarms must be a \$CIMBASIC alarm type for all details, including the alarm message (on page 811), to display correctly in an Alarm Viewer. Point alarms are not \$CIMBASIC alarms. As a result, there are limitations and</p>

		guidelines (on page 812) to be aware of if those alarm IDs are used in the script.	
ResourceId\$		String . The Resource ID to generate the alarm against. Used to control routing of the alarm.	
Message\$ (on page 811)		String . The generated alarm message to display. Note: This string is substituted into the first variable field (on page 811) of the alarm's configured message.	
DateTime		The DateTime parameter depends on the script type.	
		CimBasic	Date Variant The <code>Date</code> and <code>Now</code> functions return the Date Variant type.
		.NET C#	System.DateTime type
		VB .NET	System.DateTime type
IsUTC		BOOLEAN Whether or not the passed in timestamp is UTC.	
		TRUE	The Date-Time parameter is a UTC timestamp
		FALSE	The Date-Time parameter is not a UTC timestamp
		 Note: If you do not use UTC time, you will be responsible for making sure your system's Time Zone settings, including DST, are properly set.	
UserId\$		String (optional). The User ID that generated the alarm.	

	RefId\$	String (optional). A Reference ID used to distinguish identical alarms.
	Master	BOOLEAN (optional). By default on a computer with Server Redundancy, alarms sent by the standby computer's Event Manager are ignored. To allow an alarm to be generated from a script on a standby computer, set Master to True.
Cim- Ba- sic Ex- am- ple 1	<pre>'This example displays the syntax. Sub Main() theDate = Now() AlarmGenerateEx "PROJECT01","ALARM501","\$SYSTEM","Device 501 needs attention.", theDate, FALSE End Sub</pre>	
Ex- am- ple 2	<pre>'This example displays time in microseconds. Sub Main() TheDate = #2012/1/12 10:49:0# + 0.000002 AlarmGenerateEx "FORSHOW","MYALARM","\$MAC_FR","Hello", TheDate, true End Sub</pre>	
.NET C# Ex- am- ple 1	<pre>//This example displays the syntax. public void Main() { DateTime dt = new DateTime(2012, 06, 18, 2, 5, 5); Cimplicity.AlarmGenerateEx("TESTER","TESTALARMGEN","\$SYSTEM","csAG Test",dt, true)}</pre>	
Ex- am- ple 2	<pre>//This example displays time in microseconds. public void Main() { DateTime dt = new DateTime(2012, 7, 1, 0,0,0,123); // Add One extra millisecond + a few Nano100seconds (10000 milliseconds in a Nano100Seconds) dt = dt.AddTicks(10100); Cimplicity.AlarmGenerateEx("FORSHOW","MYALARM","\$MAC_FR",".net", dt, false); }</pre>	
VB.NET Ex-	<pre>'This example displays the syntax. Public Sub Main() Dim DT1 As DateTime</pre>	

am- ple	<pre>DT1 = New DateTime(2012,7,1,15,30,22,123) Cimplicity.AlarmGenerateEx("ALARMGENERATEUPDATE","CB1","\$SYSTEM","Test VB alarm", DT1, False) End Sub</pre>
------------	---

Guidelines: AlarmGenerateEx and AlarmUpdateEx

- Message\$ limitations and guidelines.
- Non-Point alarm requirements.
- Point alarm guidelines.

Note: Guidelines also apply to [AlarmGenerate \(on page 806\)](#) and [AlarmUpdate \(on page 813\)](#) .

Message\$ Limitations and Guidelines

Messages that display in the Alarm Viewer draw from the following sources and have the following limitations.

The message, which is a string, is substituted into the first variable field of the alarm's configured message.

<p>Message: User-defined alarm The substituted string will be the first %s in the Alarm Definition dialog box>Alarm Message field.</p>	
<p>Message : Point alarm ID The substituted string will be the first variable field (%VAL, %ID) in an Alarm Definition dialog box (or Point Properties dialog box)>Alarm Message field. However, if a point alarm ID is used in an <code>AlarmGenerateEx</code> or AlarmUpdateEx (on page 816) script, because the alarm is not a \$CIM-BASIC alarm, the message will most likely not display as you would expect. Examples The entry in the Alarm Message field includes text and more than one variable <code>POINT01 is %VAL : %STATE</code> If the code:</p>	
<p>Does not include a message</p>	<ul style="list-style-type: none"> • Text from the field will display; Variable values will not display. • The first variable position will be blank; BAD FIELD might display for the other variables. <p style="text-align: right;"><code>POINT01 is :BAD FIELD</code></p>

Does include a message "Point in alarm state."	<ul style="list-style-type: none"> • Text will display; the message in the code will display in the first variable position. • BAD FIELD might display for other variables. <pre>POINT01 is Point in alarm state. BAD FIELD</pre>
--	---

Non-Point Alarm Requirements

The alarm definition (in an Alarm Definition dialog box) for a non-point alarm must include the following values.

Alarm type	\$CIMBASIC alarm.
Alarm message	%s

Point Alarm Guidelines

When an alarm is generated using a point alarm ID, the alarm that is generated is actually no longer a point alarm.

However, like its point alarm counterpart, it also is not a \$CIMBASIC alarm.

- The alarm [message \(on page 811\)](#) may not display correctly.
- A unique alarm in CIMPLICITY is defined by the Alarm ID, Resource ID and Reference ID combination.

Each unique alarm can be displayed as a distinct entry in the Alarm Viewer.

If the actual point alarm is in alarm state and displays in the Alarm Viewer, using the same alarm ID in:

- `AlarmGenerateEx` will generate a separate alarm from the point alarm.
- `AlarmUpdateEx (on page 816)` will acknowledge and/or reset the actual alarm depending on the command(s).

Note: If only the generated alarm is listed `AlarmUpdateEx (on page 816)` will acknowledge and/or reset that alarm. Non-unique alarms are stacked, so that the user only sees the most recent occurrence. In general, the Resource ID is used to control the routing of alarms to users. The Reference ID is used by an application to distinguish between different instances of the same alarm.

AlarmUpdate (statement)

Syntax	AlarmUpdate Project\$, AlarmId\$, ResourceId\$, Action% [, AlarmMessage\$ [, UserId\$ [, RefId\$]]]	
Description	To update a currently generated alarm. The alarm being updated may be of any alarm type. However, if the AlarmMessage\$ is specified, it must be an alarm with an alarm type of \$CIM-BASIC.	
Comments	Parameter	Description
	Project\$	String . The project to generate the alarm on, an empty string "" indicates the current project
	Alarm-Id\$	String . The ID of the Alarm. Must be a valid alarm.
	Resource-Id\$	String . The Resource ID to generate the alarm against. Used to control routing of the alarm.
	Action%	Integer . Indicates whether to acknowledge or reset the alarm. Use the manifest constants <code>AM_ACKNOWLEDGED</code> and <code>AM_RESET</code> to perform an acknowledgment and a reset. By default, on a computer with Server Redundancy, alarm updates from the standby computer's Event Manager are ignored. To acknowledge or reset an alarm on the active computer from the standby computer, use <code>AM_ACKNOWLEDGED_M</code> or <code>AM_RESET_M</code> to override the default behavior.
	Alarm-Mes-sage\$	String . (optional). The update alarm message to display. Note: This string is substituted into the first variable field of the Alarm's message. For a user-defined alarm message, this will be the first %s field in the message. For a point alarm message, it will be the first variable field (%VAL, %ID, etc.) in the alarm message. For this reason, it is not recommended that you use the AlarmMessage\$ field when updating point alarms.
	UserId\$	String . (optional). The User ID which generated the alarm.
	RefId\$	String . A Reference ID used to distinguish between identical alarms. The Reference ID needs to match the Reference ID of the generated alarm. If the alarm was generated without a Reference ID, then this field can be omitted from the call.

<p>Example</p>	<pre> Sub Main() a\$ = time\$ AlarmUpdate "BCEDEMO", "MY_ALARM_1", "\$SYSTEM", x, _ "Electrical Bus 1 " & a\$ AlarmUpdate "BCEDEMO", "MY_ALARM_2", "RESOURCE_1", x, _ "Multiple Instance for each resource " & a\$ AlarmUpdate "BCEDEMO", "MY_ALARM_2", "RESOURCE_2", x, _ "Multiple Instance for each resource " & a\$ AlarmUpdate "BCEDEMO", "MY_ALARM_2", "RESOURCE_3", x, _ "Multiple Instance for each resource " & a\$ AlarmUpdate "BCEDEMO", "MY_ALARM_3", "RESOURCE_1", x, _ "Multiple Instances for RefIf " & a\$, "", "1" AlarmUpdate "BCEDEMO", "MY_ALARM_3", "RESOURCE_1", x, _ "Multiple Instances for RefIf " & a\$, "", "2" AlarmUpdate "BCEDEMO", "MY_ALARM_3", "RESOURCE_1", x, _ "Multiple Instances for RefIf " & a\$, "", "3" End Sub </pre>
<p>See Also</p>	<p>AlarmGenerate (on page 806) (statement)</p>
<p>Note</p>	<p>When updating an alarm, the AlarmId\$, ResourceId\$ and RefId\$ must match exactly to the alarm to be updated; if they do not match, the alarm will not be updated. When updating a point alarm, the RefId\$ is always the Point ID (which is also the Alarm ID).</p> <p>Guidelines that apply to AlarmUpdateEx (on page 819) also apply to <code>AlarmUpdate</code>.</p>


AlarmUpdateCA (statement)

<p>Syntax</p>	<pre> AlarmUpdate Project\$, AlarmId\$, ResourceId\$, Action% ,caObj [, AlarmMessage\$ [, UserId\$ [,RefId\$]]] </pre>	
<p>Description</p>	<p>To update a currently generated Change approval alarm. The alarm being updated may be of any alarm type. However, if the <code>AlarmMessage\$</code> is specified, it must be an alarm with an alarm type of <code>\$CIMBASIC</code>.</p>	
<p>Comments</p>	<p>Parameter</p>	<p>Description</p>


Project \$	String. The project to generate the alarm on, an empty string "" indicates the current project
Alarm-Id\$	String. The ID of the Alarm. Must be a valid alarm.
Re-source-Id\$	String. The Resource ID to generate the alarm against. Used to control routing of the alarm.
Ac-tion%	Integer. Indicates whether to acknowledge or reset the alarm. Use the manifest constants <code>AM_ACKNOWLEDGED</code> and <code>AM_RESET</code> to perform an acknowledgment and a reset. By default, on a computer with Server Redundancy, alarm updates from the standby computer's Event Manager are ignored. To acknowledge or reset an alarm on the active computer from the standby computer, use <code>AM_ACKNOWLEDGED_M</code> or <code>AM_RESET_M</code> to override the default behavior.
caObj	Object of type <code>CimAlmChangeApprovalData</code> . It contains Change Approval information.
Alarm-Mes-sage\$	String. (optional). The update alarm message to display. Note: This string is substituted into the first variable field of the Alarm's message. For a user-defined alarm message, this will be the first %s field in the message. For a point alarm message, it will be the first variable field (%VAL, %ID, etc.) in the alarm message. For this reason, it is not recommended that you use the AlarmMessage\$ field when updating point alarms.
UserId \$	String. (optional). The User ID which generated the alarm.
RefId\$	String. A Reference ID used to distinguish between identical alarms. The Reference ID needs to match the Reference ID of the generated alarm. If the alarm was generated without a Reference ID, then this field can be omitted from the call.
Exam-ple	<pre> Sub Main() Dim obj As New CimAlmChangeApprovalData obj.PerformerUserid = "OPERATOR" obj.PerformerPassword = "" obj.PerformerComment= "bool=1 from BCE" obj.VerifierUserid = "MANAGER" obj.VerifierPassword = "" obj.VerifierComment= "bool=1 from BCE" </pre>

	<pre>AlarmUpdateCa"ESIGDEMO", "CA_TESTPOINT", "\$MAC_FR", AM_ACKNOWLEDGED, CAobj, "CA_TESTPOINT", "CA_TESTPOINT", "CA_TESTPOINT" End Sub</pre>
See Also	AlarmGenerate (on page 806) (statement)
Note	When updating an alarm, the AlarmId\$, ResourceId\$ and RefId\$ must match exactly to the alarm to be updated; if they do not match the alarm will not be updated. When updating a point alarm, the RefId\$ is always the Point ID (which is also the Alarm ID).

AlarmUpdateEx (statement)

Syntax	AlarmUpdateEx Project\$, AlarmId\$, ResourceId\$, Action%, DateTime, IsUTC [, AlarmMessage\$ [, UserId\$ [, RefId\$]]]	
	Parameter	Description
	Project\$	String. The project to update the alarm on. An empty string "" indicates the current project.
	AlarmId\$	<p>String. The ID of a non-point or point Alarm that is listed in the right-pane of the Workbench>Alarms section.</p> <div style="border: 1px solid #0070C0; border-radius: 10px; padding: 10px; margin: 10px 0;">  Note: </div> <p>Non-point alarms must be a <code>\$CIMBASIC</code> alarm type for all details, including the alarm message (on page 819), to display correctly in an Alarm Viewer. Point alarms are not <code>\$CIMBASIC</code> alarms. As a result, there are limitations and guidelines (on page 820) to be aware of if those alarm IDs are used in the script.</p>
	ResourceId\$	String. The Resource ID to update the alarm against. Used to control routing of the alarm.
	Action	<p>Integer. Indicates whether to acknowledge or reset the alarm. Use the following constants to perform an acknowledgment and a reset.</p> <ul style="list-style-type: none"> • <code>AM_ACKNOWLEDGED</code> • <code>AM_RESET</code>

	<p>Server Redundancy: By default, on a computer with Server Redundancy, alarm updates from the standby computer's Event Manager are ignored. To acknowledge or reset an alarm on the active computer from the standby computer, use either of the following to override the default behavior.</p> <ul style="list-style-type: none"> • AM_ACKNOWLEDGED_M • AM_RESET_M 						
DateTime	The DateTime parameter depends on the script type.						
	<table border="1"> <tr> <td>CimBasic</td> <td>Date Variant The <code>Date</code> and <code>Now</code> functions return the Date Variant type.</td> </tr> <tr> <td>.NET C#</td> <td>System.DateTime type</td> </tr> <tr> <td>VB .NET</td> <td>System.DateTime type</td> </tr> </table>	CimBasic	Date Variant The <code>Date</code> and <code>Now</code> functions return the Date Variant type.	.NET C#	System.DateTime type	VB .NET	System.DateTime type
CimBasic	Date Variant The <code>Date</code> and <code>Now</code> functions return the Date Variant type.						
.NET C#	System.DateTime type						
VB .NET	System.DateTime type						
IsUTC	BOOLEAN Whether or not the passed in timestamp is UTC.						
	<table border="1"> <tr> <td>TRUE</td> <td>The Date-Time parameter is a UTC timestamp</td> </tr> <tr> <td>FALSE</td> <td>The Date-Time parameter is not a UTC timestamp</td> </tr> </table>	TRUE	The Date-Time parameter is a UTC timestamp	FALSE	The Date-Time parameter is not a UTC timestamp		
TRUE	The Date-Time parameter is a UTC timestamp						
FALSE	The Date-Time parameter is not a UTC timestamp						

		 Note: If you do not use UTC time, you will be responsible for making sure your system's Time Zone settings, including DST, are properly set.
	Message\$ (on page 819)	String. The update alarm message to display. Note: This string is substituted into the first variable field (on page 819) of the Alarm's message.
	UserId\$	String (optional). The User ID that updated the alarm.
	RefId\$	String (optional). A Reference ID used to distinguish identical alarms.
	Master	BOOLEAN (optional). By default on a computer with Server Redundancy, alarms sent by the standby computer's Event Manager are ignored. To allow an alarm to be generated from a script on a standby computer, set Master to True.
Cim- Ba- sic Ex- am- ple	<pre>'This example displays the syntax. Sub Main() theDate = Now() AlarmUpdateEx "TESTER", "ALARM501", "\$SYSTEM", AM_ACKNOWLEDGED, theDate, FALSE "Device 501: Responded." End Sub</pre>	
.NET C# Ex- am- ple	<pre>//This example displays the syntax. public void Main() { DateTime dt = new DateTime(2012, 06, 18, 2, 5, 5); Cimplicity.AlarmUpdateEx("TESTER", "TESTALARMGEN", "\$SYSTEM", Cimplicity.AM_ACKNOWLEDGED, dt, true, "csAG Test"); }</pre>	
VB.NET Ex- am- ple	<pre>'This example displays the syntax. Public Sub Main() Dim DT2 As DateTime DT2 = New DateTime (2012,7,10,20,20,30,789) Cimplicity.AlarmUpdateEx("ALARMGENERATEUPDATE", "CB1", "\$SYSTEM", Cimplicity.AM_RESET + Cimplicity.AM_ACKNOWLEDGED, DT2, True, "Updated", "OPERATOR") End Sub</pre>	

Guidelines: AlarmGenerateEx and AlarmUpdateEx

- Message\$ limitations and guidelines.
- Non-Point alarm requirements.
- Point alarm guidelines.

Note: Guidelines also apply to [AlarmGenerate \(on page 806\)](#) and [AlarmUpdate \(on page 813\)](#).

Message\$ Limitations and Guidelines

Messages that display in the Alarm Viewer draw from the following sources and have the following limitations.

The message, which is a string, is substituted into the first variable field of the alarm's configured message.

<p>Message: User-defined alarm The substituted string will be the first %s in the Alarm Definition dialog box>Alarm Message field.</p>	
<p>Message : Point alarm ID The substituted string will be the first variable field (%VAL, %ID) in an Alarm Definition dialog box (or Point Properties dialog box)>Alarm Message field. However, if a point alarm ID is used in an AlarmGenerateEx (on page 808) or AlarmUpdateEx script, because the alarm is not a \$CIM-BASIC alarm, the message will most likely not display as you would expect. Examples The entry in the Alarm Message field includes text and more than one variable POINT01 is %VAL : %STATE If the code:</p>	
<p>Does not include a message</p>	<ul style="list-style-type: none"> • Text from the field will display; Variable values will not display. • The first variable position will be blank; BAD FIELD might display for the other variables. <p>POINT01 is :BAD FIELD</p>
<p>Does include a message "Point in alarm state."</p>	<ul style="list-style-type: none"> • Text will display; the message in the code will display in the first variable position. • BAD FIELD might display for other variables. <p>POINT01 is Point in alarm state. BAD FIELD</p>

Non-Point Alarm Requirements

The alarm definition (in an Alarm Definition dialog box) for a non-point alarm must include the following values.

Alarm type	\$CIMBASIC alarm.
Alarm message	%s

Point Alarm Guidelines

When an alarm is generated using a point alarm ID, the alarm that is generated is actually no longer a point alarm.

However, like its point alarm counterpart, it also is not a \$CIMBASIC alarm.

- The alarm [message \(on page 819\)](#) may not display correctly.
- A unique alarm in CIMPLICITY is defined by the Alarm ID, Resource ID and Reference ID combination.

Each unique alarm can be displayed as a distinct entry in the Alarm Viewer.

If the actual point alarm is in alarm state and displays in the Alarm Viewer, using the same alarm ID in:

- [AlarmGenerateEx \(on page 808\)](#) will generate a separate alarm from the point alarm.
- [AlarmUpdateEx](#) will acknowledge and/or reset the actual alarm depending on the command(s).

Note: If only the generated alarm is listed [AlarmUpdateEx](#) will acknowledge and/or reset that alarm.

- Non-unique alarms are stacked, so that the user only sees the most recent occurrence. In general, the Resource ID is used to control the routing of alarms to users.
- The Reference ID is used by an application to distinguish between different instances of the same alarm.

ChangePassword (statement)

Syntax	ChangePassword Project\$, OldPassword\$, NewPassword\$
Description	To change a password for a currently logged in user on a specified project.

Comments	Parameter	Description
	Project\$	String. The project to change the password on. An empty string indicates the current default project.
	OldPassword\$	String. The old password of the user.
	NewPassword\$	String. The new password of the user.
Example	<pre>Sub Main() ChangePassword "CIMPDEMO", "OLDPASS", "NEWPASS" End Sub</pre>	
Note	The user must be logged into the specified project or the function will fail.	

CimChangeApprovalData (Object)

Overview	The <code>CimChangeApprovalData</code> object contains information about Change Approval information (e.g. Performer and Verifier User ID, Password and Comments for Point Operations).
Example	<pre>Dim obj As New CimAlmChangeApprovalData obj.PerformerUserid = "OPERATOR" obj.PerformerPassword = "" obj.PerformerComment= "bool=1 from BCE" obj.VerifierUserid = "MANAGER" obj.VerifierPassword = "" obj.VerifierComment= "bool=1 from BCE"</pre>

CimEMAlarmEvent.AlarmID (property, read)

Syntax	AlarmEvent.AlarmId
Description	String. Returns the Alarm ID of the Alarm that triggered the event.
Example	<pre>Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent()</pre>

```

PointSet "LAST_ALARM_ID", AlarmEvent.AlarmID

End if

End Sub
    
```

CimEMAlarmEvent.FinalState (property, read)

Syntax	AlarmEvent.FinalState
Description	Integer. Returns the final state of the alarm after the requested action. For example, if the user acknowledged the alarm and the deletion requirements for the alarm only require acknowledgement then the final state would be AM_DELETED.
	<p>Valid States are :</p> <ul style="list-style-type: none"> • AM_GENERATED • AM_ACKNOWLEDGED • AM_RESET • AM_DELETED
Example	<pre> Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() If AlarmEvent.FinalState = AM_ACKNOWLEDGED then PointSet "ALARM_MESSAGE", "Alarm is Acknowledged" End if End Sub </pre>

CimEMAlarmEvent.GenTime (property, read)

Syntax	AlarmEvent.GenTime
Description	Date. Returns the day and time the alarm was generated.
Example	<pre> Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() PointSet "TEXT_ALARM_GEN_TIME", cstr(AlarmEvent.GenTime) End Sub </pre>

```
End if
End Sub
```

CimEMAlarmEvent.Message (property, read)

Syntax	AlarmEvent.Message
Description	String. Returns the text of the Alarm Message of the alarm that triggered the event.
Example	<pre>Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() PointSet "LAST_ALARM_MESSAGE", AlarmEvent.Message End if End Sub</pre>

CimEMAlarmEvent (object)

Overview	The CimEMAlarmEvent object provides information for scripts invoked from an alarm event.
Example	<pre>Dim alarmEvent As CimEmAlarmEvent Set alarmEvent = CimGetEMEvent().AlarmEvent() PointSet "ALARM_MESSAGE", alarmEvent.Message</pre>
Note	CimEMAlarmEvent can only be used from the Event Manager. It is not valid in CimView/Cim-Edit.

CimEMAlarmEvent.PrevState (property, read)

Syntax	AlarmEvent.PrevState
Description	<p>Integer. Returns the previous state of the alarm. Valid States are :</p> <ul style="list-style-type: none"> • AM_GENERATED • AM_ACKNOWLEDGED • AM_RESET • AM_DELETED

Example	<pre> Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() If AlarmEvent.PrevState = AM_ACKNOWLEDGED then PointSet "ALARM_PREVSTATE", "ACKNOWLEDGED" End if End Sub </pre>
----------------	--

CimEMAlarmEvent.RefID (property, read)

Syntax	AlarmEvent.RefID
Description	String. Returns the Reference ID of the alarm that triggered the event.
Example	<pre> Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() PointSet "LAST_ALARM_REF_ID", AlarmEvent.RefID End if End Sub </pre>

CimEMAlarmEvent.ReqAction (property, read)

Syntax	AlarmEvent.ReqAction
Description	Integer. Returns the action requested on the alarm. For example, if the user had acknowledged the alarm in the Alarm Viewer the requested action would be AM_ACKNOWLEDGED.
Example	<pre> Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() If AlarmEvent.ReqAction = AM_ACKNOWLEDGED then PointSet "ALARM_MESSAGE", "Alarm has been Acknowledged" End if End Sub </pre>

CimEMAlarmEvent.ResourceID (property, read)

Syntax	AlarmEvent.ResourceID
Description	String. Returns the Resource ID of the alarm that triggered the event.
Example	<pre> Sub Main() Dim AlarmEvent as CimEmAlarmEvent Set AlarmEvent = CimGetEMEvent().AlarmEvent() PointSet "LAST_ALARM_RESOURCE_ID", AlarmEvent.ResourceID End if End Sub </pre>

CimEMEvent.ActionID (property, read)

Syntax	Event.ActionID
Description	String. Returns the Action ID that is a running the script.
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() PointSet "LAST_ACTION_ID", event.ActionID End Sub </pre>

CimEMEvent.AlarmEvent (function)

Syntax	Event.AlarmEvent
Description	Returns CimEMAlarmEvent. Returns the Alarm Event object that triggered the action, or empty if action was not triggered by an alarm.
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() If event.Type = EM_ALARM_GEN then Dim alarmEvent as CimEMAlarmEvent Set AlarmEvent = event.AlarmEvent() </pre>

```

        ' Process the alarm

    End If

End Sub
    
```

CimEMEvent.EventID (property, read)

Syntax	Event.EventID
Description	String. Returns the EventID that triggered the event.
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() PointSet "LAST_EVENT_ID", event.EventId End Sub </pre>

CimEMEvent (object)

Overview	An object used by the Event Manager to hold information about the event that triggered the action.
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() PointSet "LAST_EVENT_ID", event.EventId End Sub </pre>
Note	CimEMEvent can only be used from the Event Manager. It is not valid in CimView/CimEdit.

CimEMEvent.ObjectID (property, read)

Syntax	Event.ObjectID
Description	String. If the script is invoked from an object event, the Object ID invoking the action is returned. If the script is invoked from a non-object event, an empty string is returned

Ex- am- ple	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() PointSet "LAST_OBJECT_ID", event.ObjectID End Sub </pre>
-------------------	---

CimEMEvent.PointEvent

Syntax	Event.PointEvent
De- scrip- tion	Returns CimEMPointEvent. Returns the Point Event object that triggered the action, or empty if action was not triggered by point event.
Exam- ple	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() Dim pointEvent as CimEMPointEvent Set pointEvent = event.PointEvent() End Sub </pre>

CimEMEvent.TimeStamp (property, read)

Syntax	Event.TimeStamp
Description	Date. Returns the Time Stamp at which the event occurred.
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() PointSet "LAST_EVENT_TIME", cstr(eent.TimeStamp) End Sub </pre>

CimEMEvent.Type (property, read)

Syntax	Event.Type
--------	-------------------

Description	Integer. Returns the type of event that triggered the action . Valid values are:	
	EM_ALARM_GEN	Alarm Generated
	EM_ALARM_ACK	Alarm Acknowledged
	EM_ALARM_RST	Alarm Reset
	EM_ALARM_DEL	Alarm Deleted
	EM_POINT_CHANGE	Point Changed
	EM_POINT_UNAVAIL	Point Unavailable
	EM_POINT_EQUALS	Point Equals
	EM_POINT_UPDATE	Point Updated
	EM_POINT_TRANS_HIGH	Point Transition to High
	EM_POINT_TRANS_LOW	Point Transition to Low
	EM_TIMED	Timed Event
	EM_RUN_ONCE	Run Once
	EM_TRIGGERED	Externally trigged by BCEUI or Action Calendar
	Consult the Event Editor documentation for more details.	
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() If event.Type = EM_ALARM_GEN then Dim alarmEvent as CimEMAlarmEvent Set AlarmEvent = event.AlarmEvent() ' Process the alarm End If End Sub </pre>	

CimEMPointEvent.Id

Syntax	PointEvent.Id
--------	----------------------

Description	String. Returns the Point ID of the point that triggered the event.
Example	<pre> Sub Main() Dim PointEvent as CimEmPointEvent Set PointEvent = CimGetEMEvent().PointEvent() ' perform processing ' reset the event point to 0 PointSet PointEvent.Id, 0 End Sub </pre>
Note	CimEMPointEvent can only be used from the Event Manager. It is not valid in CimView/ CimEdit

CimEMPointEvent (object)

Overview	An Event Manager Object used to contain information about a Point Event
Example	<pre> Sub Main() Dim PointEvent as CimEmPointEvent Set PointEvent = CimGetEMEvent().PointEvent() ' perform processing ' reset the event point to 0 PointSet PointEvent.Id, 0 End Sub </pre>

CimEmPointEvent.Quality (property, read)

Syntax	CimEMPointEvent.Quality
Description	Long. Returns the 16-bit quality mask for the point that triggered the event.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() X = p.Quality End Sub </pre>

CimEmPointEvent.QualityAlarmed (property, read)

Syntax	CimEMPointEvent.QualityAlarmed
Description	Boolean. Returns TRUE if the point that triggered the event is in alarm, FALSE otherwise.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() if p.QualityAlarmed then DoSomething End If End Sub </pre>

CimEmPointEvent.QualityAlarms_Enabled (property, read)

Syntax	CimEMPointEvent.QualityAlarms_Enabled
Description	Boolean. Returns TRUE if alarming for the point that triggered the event is enabled, FALSE otherwise.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() if p.QualityAlarms_Enabled then DoSomething End If End Sub </pre>

CimEmPointEvent.QualityDisable_Write (property, read)

Syntax	CimEMPointEvent.QualityDisable_Write
Description	Boolean. Returns TRUE if setpoints have been disabled for the point that triggered the event, FALSE otherwise.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() </pre>

```

    if p.QualityDisable_Write Then
        DoSomething
    End If
End Sub

```

CimEmPointEvent.QualityLast_Upd_Man (property, read)

Syntax	CimEMPointEvent.QualityLast_Upd_Man
Description	Boolean. Returns TRUE if the value of the point that triggered the event came from a manual update rather than a device read.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() If p.QualityLast_Upd_Man then DoSomething End If End Sub </pre>

CimEmPointEvent.QualityManual_Mode (property, read)

Syntax	CimEMPointEvent.QualityManual_Mode
Description	Boolean. Returns TRUE if the point that triggers the event was in Manual Mode, otherwise FALSE.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() if p.QualityManual_Mode then ProcessManualMode End if End Sub </pre>

CimEmPointEvent.QualityIs_In_Range (property, read)

Syntax	CimEMPointEvent.QualityIs_In_Range
Description	Boolean. Returns TRUE if the value of the point that triggered the event is in range, FALSE if the point is out of range. When a point is out of range its value is unavailable.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() if p.QualityIs_In_Range = FALSE then DoSomething End If End Sub </pre>

CimEmPointEvent.QualityStale_Data (property, read)

Syntax	CimEMPointEvent.QualityStale_Data
Description	Boolean. Returns TRUE if the value of the point that triggered the event is stale, otherwise FALSE.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() if p.QualityStale_Data = TRUE DoSomething End If End Sub </pre>

CimEmPointEvent.QualityIs_Available (property, read)

Syntax	CimEMPointEvent.QualityIs_Available
Description	Boolean. Returns TRUE if the value of the point that triggered the event is available, FALSE if the value is unavailable.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() </pre>

```

if p.QualityIs_Available = FALSE then
    DoSomething
End If
End Sub
    
```

CimEMPointEvent.State (property, read)

Syntax	PointEvent.State
Description	Integer. Returns the state of the point. Can be used to determine if the point is available. See Point.State for a complete description of states.
Example	<pre> Sub Main() Dim PointEvent as CimEmPointEvent Set PointEvent = CimGetEMEvent().PointEvent() If PointEvent.State = CP_UNAVAILABLE THEN LogStatus CIM_FAILURE,"Main()", _ "Point " & Point.Id & "is unavailable" end End if End Sub </pre>

CimEMPointEvent.TimeStamp (property, read)

Syntax	PointEvent.TimeStamp
Description	Date. Returns the date and time of the point change that triggered the event.)
Example	<pre> Sub Main() Dim PointEvent as CimEmPointEvent Set PointEvent = CimGetEMEvent().PointEvent() PointSet "LAST_EVENT_TIME", cstr(PointEvent.TimeStamp) End Sub </pre>

CimEmPointEvent.UserFlags (property, read)

Syntax	CimEMPointEvent.UserFlags
Description	Long. Returns the value of the 16-bit user defined flags for the point that triggered the event.
Example	<pre> Sub Main() Dim p as new CimEMPointEvent Set p = CimGetEmEvent().PointEvent() X = p.UserFlags End Sub </pre>

CimEMPointEvent.Value (property, read)

Syntax	PointEvent.Value
Description	Variant. Returns the value of the point that triggered the event.
Example	<pre> Sub Main() Dim PointEvent as CimEmPointEvent Set PointEvent = CimGetEMEvent().PointEvent() PointSet "OUTPUT_POINT", PointEvent.Value + 100 End Sub </pre>

CimGetEMEvent (function)

Syntax	CimGetEMEvent()
Description	Returns a CimEMEvent object. A function to return the event object that causes the action to run. Only valid from Event Manager.
Example	<pre> Sub Main() Dim event as CimEMEvent Set event = CimGetEMEvent() PointSet "LAST_EVENT_TIME", cstr(event.TimeStamp) End Sub </pre>
Note	<code>CimGetEMEvent</code> can only be used from the Event Manager. It is not valid in CimView/CimEdit.

CimIsMaster (function)

Syn- tax	CimIsMaster
De- scrip- tion	In a computer with Server Redundancy, to determine if the computer is operating in Active or Standby mode. This function returns TRUE if the computer is currently the active computer. This function returns FALSE if the computer is currently the standby.
Ex- am- ple	<pre> Sub Main() If CimIsMaster then MoveCrane End if End Sub </pre>

CimLogin (statement)

Syn- tax	CimLogin project\$	
De- scrip- tion	Initiates a login for the specified project. Similar in effect to selecting login from the Login Panel. Only valid when the user is actively using points or viewing alarms from the project, otherwise it has no effect. Initiating a login will cause the CIMPLICITY login box to be displayed.	
Com- ments	Parameter	Description
	project\$	String . The project to login to.
Exam- ple	<pre> Sub Main() CimLogin "CIMPDEMO" End Sub </pre>	

CimLogout (statement)

Syn- tax	CimLogout project\$
De- scrip- tion	Logs the user out of the specified project. Similar in effect to selecting logout from the Login Panel. When the user is logged out of the project, all points from the project will be unavailable

	and no alarm information will be available. If the user is not logged into the project, the call has no effect.	
Comments	Parameter	Description
	project\$	String. The project to logout of.
Example	<pre>Sub Main() CimLogout "CIMPDEMO" End Sub</pre>	

CimProjectData.Attributes (property, read/write)

Syntax	CimProjectData.Attributes
Description	String. The list of attributes, separated by commas, of the entity to return for each item matching the filter criteria. The Attribute IDs are case sensitive and must be entered in the case documented in CimProjectData.Entity .
CimBasic Example	<pre>Dim d as new CimProjectData d.Attributes = "POINT_ID,RESOURCE_ID,DESCRIPTION"</pre>
.Net Example	<pre>CimProjectData cpd = new CimProjectData(); cpd.Attributes = "POINT_ID,RESOURCE_ID";</pre>

CimProjectData.Filters (property, read/write)

Syntax	CimProjectData.Filters
Description	String. The filter set to be used to determine which items to return. Each filter contains an Attribute ID and Value pair. You can use "*" and "?" as wildcard characters. The filters are documented in CimProjectData.Entity . Filters must be in uppercase even when matching against lowercase data.
Example	<pre>Dim d as new CimProjectData d.Filters = "POINT_ID=P*",DEVICE_ID=TESTP?C"</pre>

CimProjectData.GetNext (function)

Syntax	CimProjectData.GetNext(p1\$ [,p2\$ [,p3\$...]) as Boolean	
Description	This function returns the specified attributes for the next item that matches the filter criteria. If a record is found, a value of TRUE is returned, otherwise a value of FALSE is returned. The function takes a variable number (20 maximum) of string parameters. The values returned into the parameters are defined by the attributes specified for the object.	
Comments	Parameter	Description
	p1\$	String . First attribute for the object.
	:	:
	p20\$	String . Twentieth attribute for the object.
Example 1	<p>The following sample script returns all the data items for the PID1 object.</p> <pre> Sub Main() Dim browse as new CimProjectData Browse.Project = "MY_PROJ" Browse.Entity = "OBJECT_INF" Browse.Attributes = DATA_ITEM" Browse.Filters = "OBJECT_ID=PID1" Dim dataItem as String Top: If Browse.GetNext(dataItem) = False then end MsgBox dataitem Goto top End Sub </pre>	
Example 2	<p>The following sample script returns all points for a device:</p> <pre> Sub Main() Dim browse as new CimProjectData Browse.Project = "MY_PROJ" Browse.Entity = "POINT" Browse.Attributes = "POINT_ID,RESOURCE_ID" Browse.Filters = "DEVICE_ID=PLC1" Top: </pre>	

```

If Browse.GetNext(p$,r$) = False then end

Msgbox "Point Id " & p$ & " Resource id " & r$

Goto top

End Sub
    
```

CimProjectData.Entity (property, read/write)

Syntax	CimProjectData.Entity
Description	String. The entity to obtain data for. Below is a list of the available entities and their attributes
Example	Dim d as New CimProjectData d.Entity = "POINT"

Entity List

ACTION	MEASSYSTEM	PORT
ALARM_BLK_GROUP	MEASUNIT	PROJECTS
ALARM_CLASS	OBJECT	PROTOCOL
ALARM_DEF	OBJECT_INF	RESOURCE
AMLPL	POINT	ROLE
CLASS	POINT_ALSTR	SSPC
CLIENT	POINT_DISP	SYS_PARMS
DEVICE	POINT_ENUM	UAFSETS
EVENT	POINT_ENUM_FLD	USER
EVENT_ACTION	POINT_TYPE	USER_FIELDS
GLB_PARMS		

ACTION

Contains Action information

Attribute ID	Filter	Description
ACTION_ID	Yes	Action ID

ACTION_TYPE	No	Action Type
POINT_ID	No	Point ID targeted by the action
PT_VAL	No	Point value
PROC_OF_SRCPT	No	Source point,

ALARM_BLK_GROUP

Contains Alarm Blocking Group Information.

Attribute ID	Filter	Description
BLOCK_GROUP_ID	Yes	Alarm Blocking Group ID
DESCRIPTION	Yes	Description of the group.
PEER_BLOCK	Yes	Blocking Mode: If you select Peer Blocking mode, only the first alarm of a set of alarms with equal priority displays for that group.

ALARM_CLASS

Contains Alarm Class information.

Attribute ID	Filter	Description
CLASS_ID	Yes	Class ID
CLASS_TITLE	Yes	Class title
CLASS_ORDER	No	Class order
CLASS_ALARM_FG	No	The foreground color to use for points of this class that are in alarm state.
CLASS_ALARM_BG	No	The background color to use for points of this class that are in alarm state.
CLASS_NORMAL_FG	No	The foreground color to use for points of this class that are in normal state.

CLASS_NORMAL_BG	No	The background color to use for points of this class that are in normal state.
CLASS_ACK_FG	No	The foreground color to use for points of this class that are in acknowledged state.
CLASS_ACK_BG	No	The background color to use for points of this class that are in acknowledged state.
CLASS_WAVE_FILE	No	The WAV file to play from the Alarm Sound Manager.
CLASS_BEEP_FREQ	No	Frequency of beeps from the Alarm Sound Manager.
CLASS_BEEP_DURATION	No	Duration of beeps from the Alarm Sound Manager.
CLASS_BEEP_DELAY	No	Delay between beeps from the Alarm Sound Manager.
CLASS_ALARM_BLINK_RATE	No	Delay between blinks when in an Alarm state.
CLASS_ALARM_BLINK_FG	No	The foreground color to use when in an Alarm state.
CLASS_ALARM_BLINK_BG	No	The background color to use when in an Alarm state.
CLASS_NORMAL_BLINK_RATE	No	Delay between blinks when in a Normal state.
CLASS_NORMAL_BLINK_FG	No	The foreground color to use when in a Normal state.
CLASS_NORMAL_BLINK_BG	No	The background color to use when in a Normal state.
CLASS_ACK_BLINK_RATE	No	Delay between blinks when in an Acknowledged state.
CLASS_ACK_BLINK_FG	No	The foreground color to use when in an Acknowledged state.
CLASS_ACK_BLINK_BG	No	The background color to use when in an Acknowledged state.
CLASS_BEEP_NUMBER	No	The number of beeps sounded for the alarm.

ALARM_DEF

Contains Alarm information.

Attribute ID	Filter	Description
ALARM_ID	Yes	Alarm ID
CLASS_ID	Yes	Alarm Class of the alarm.
ALARM_MSG	Yes	Returns the configured alarm message on the alarm.
ALARM_TYPE_ID	Yes	Alarm Type ID of the alarm.
DESCRIPTION	Yes	Description of the alarm.

Sample Script

```

Dim objCimProjectData      As CimProjectData
Dim strOptionalProject     As String
Dim AlID                  As String
Dim AlarmMessage As String

Set objCimProjectData = New CimProjectData

objCimProjectData.Project = strOptionalProject
objCimProjectData.Entity = "ALARM_DEF"
objCimProjectData.Filters = "CLASS_ID=HIGH"
objCimProjectData.Attributes = "ALARM_ID,ALARM_MSG" 'Set the attribute to retrieve

'get The alarm info

While objCimProjectData.GetNext(AlID,AlarmMessage) = True

    MsgBox AlID

    MsgBox AlarmMessage

Wend

```

AMLPL

Contains Alarm Printer information.

Attribute ID	Filter	Description
AMLPL_NAME	Yes	Alarm printer name.
AMLPL_PORT	No	Alarm printer port.
PAGE_WIDTH	No	Page width.
PAGE_LENGTH	No	Page length.

DATE_FORMAT	No	Date format.
TIME_FORMAT	No	Time format.

CLASS

Contains Class information.

Attribute ID	Filter	Description
CLASS_ID	Yes	Class ID.
DESCRIPTION	Yes	Description of the class.

CLIENT

Contains Client information.

Attribute ID	Filter	Description
NODE_ID	Yes	Computer name.
USER_ID	No	Default User ID.
TRUSTED	No	Trusted computer.

DEVICE

Contains Device information.

Attribute ID	Filter	Description
DEVICE_ID	Yes	Device ID.
RESOURCE_ID	Yes	Resource ID for the device.
DESCRIPTION	Yes	Device description.
PORT_ID	Yes	Port ID for the device.

EVENT

Contains Event information.

Attribute ID	Filter	Description
EVENT_ID	Yes	Event ID.
EVENT_TYPE	No	Event type.
EM_ENABLED	No	Event enabled flag.
ID	No	Event source identifier.
RESOURCE_ID	No	Resource ID of the event.
PT_VAL	No	For Point Equal event, the value of the point.
SERVICE_ID	No	

EVENT_ACTION

Contains Event-Action information.

Attribute ID	Filter	Description
EVENT_ID	Yes	Event ID.
ACTION_ID	Yes	Action ID for the event.
LOG_FLAG	No	Flag indicating if the event-action is to be logged.
EA_SERVICE_ID	Yes	

GLB_PARMS

Contains Global Parameter information for the project.

Attribute ID	Filter	Description
PARAM_ID	Yes	Global Parameter ID.
PARAM_TYPE	No	Type of the global parameter.
PARAM_VALUE	No	Value of the global parameter.

MEASUNIT

Contains Measurement Unit information.

Attribute ID	Filter	Description
UNIT_ID	Yes	Identifier for the Measurement Unit.
DESCRIPTION	Yes	Description displayed for the measurement unit.
LABEL	Yes	Label displayed for the measurement unit.

MEASSYSTEM

Contains Measurement System Information

Attribute ID	Filter	Description
UNIT_ID	Yes	Identifier for the Measurement System.
DESCRIPTION	Yes	Description displayed for the measurement system.
LABEL	Yes	Label displayed for the measurement system.

OBJECT

Contains object information.

Attribute ID	Filter	Description
OBJECT_ID	Yes	Object ID.
CLASS_ID	Yes	Class ID for the object.
DESCRIPTION	Yes	Object description.

OBJECT_INF

This is a specialized entity used to extract information from a specified object. The filter for this entity is OBJECT_ID=MY_OBJECT, where MY_OBJECT is replaced with the object name you wish to read. Since the function returns specialized attribute information, only one of the attributes may be used at a time.

This entity may not be used from the Event Manager or without a specified running project.

Attribute ID	Filter	Description
DATA_ITEM	No	Returns all data items for the object. Each data item returns by a GetNext call.
ATTRIBUTE, VALUE	No	Returns the attribute for the object. If VALUE is specified, it must be the second attribute, and the value of the attribute will be returned
CLASS_ID	No	The Class ID of the object.
DEFAULT_GRAPHIC	No	Returns the name of the default graphic for the object's class. Must be specified with GRAPHICS_FILE Example <code>obj.Attributes= "GRAPHICS_FILE,DEFAULT_GRAPHIC"</code>
GRAPHICS_FILE	No	The Graphics File specified for the objects class
HELP_FILE	No	The Help File specified for the objects class

POINT

Contains Point information.

Attribute ID	Filter	Description
POINT_ID	Yes	Point ID
DEVICE_ID	Yes	Device ID for the point, where the point data originates. If the point is a global point, the device is "\$GLOBAL". If the point is an equation point, the device is \$DERIVED.
RESOURCE_ID	Yes	Resource ID of the point.
POINT_TYPE_ID	Yes	Point Type ID of the point (UINT, REAL, etc.)
DESCRIPTION	Yes	Description of the point.
DISPLAY_LIMITS_HI	No	High display limit of the point.
DISPLAY_LIMITS_LO	No	Low display limit of the point.

DISPLAY_- LIMITS	No	Low and high display limits of the point separated by a hyphen.
DISPLAY_- FORMAT	No	Display format for the point.
ELEMENTS	No	Number of array elements.
HAS_LOG	No	State of the "Log Data" checkbox on the point properties
ADDRESS	No	Device address of the point.
ADDRESS_- OFFSET	No	Address offset for the point.
HAS_EU	No	Set to 1 if the point has EU Conversion, otherwise set to 0.
ALARM_HI	No	High alarm limit for the point.
ALARM_LO	No	Low alarm limit for the point.
WARNING_- HI	No	High warning limit for the point.
WARNING_- LO	No	Low warning limit for the point.
ACCESS_- FILTER	Yes	If the point is an enterprise point, this field is set to E.
READ_- WRITE	No	If point is read/write.
MODIFIED	No	Data and time in string format that the point was last edited.
DATA_- TYPE	No	Point or SCAPL.
POINT_- CLASS	No	Point class
ORIGIN	No	Device or derived (virtual)
DATA_- LENGTH	No	Data length
NEED_UP- DATE	No	Update criteria

UNIT_ID	No	Measurement units
SET_NAME	No	Attribute set
ENUM_ID	No	Point enumeration
LEVEL	No	Security level.

POINT_ALSTR

Contains Alarm String information.

Attribute ID	Filter	Description
ALARM_STR_ID	No	Alarm String ID.
ALARM_HI_STR	No	String for Alarm High state.
ALARM_LOW_STR	No	String for Alarm Low state.
WARNING_HI_STR	No	String for Warning High state.
WARNING_LO_STR	No	String for Warning Low state.
NORMAL_STR	Yes	String for Normal state.
ALARM_HIGH_SEVERITY	No	Alarm High Severity level.
ALARM_LOWEVERITY	No	Alarm Low Severity level.
WARNING_HI_SEVERITY	No	Warning High Severity level.
WARNING_LOW_SEVERITY	No	Warning Low Severity level.
NORMAL_SEVERITY	No	Normal Severity level.

POINT_DISP

Contains Point Display information.

Attribute ID	Filter	Description
POINT_ID	Yes	Point ID.
SCREEN_ID	No	The screen associated with the point.

DISPLAY_LIM- LOW	No	The low limit for the point value display. Values below this limit will display as asterisks (**).
DISPLAY_LIM- HIGH	No	The high limit for the point value display. Values above this limit will display as asterisks (**).

POINT_ENUM

Contains Point Enumeration information.

Attribute ID	Filter	Description
ENUM_ID	Yes	Point Enumeration ID.
DESCRIPTION	Yes	Description of the enumeration.

POINT_ENUM_FLD

Contains Point Enumeration Field information.

Attribute ID	Filter	Description
ENUM_ID	Yes	Enumeration ID for the field.
VALUE	Yes	The numerical value of the enumeration.
TEXT	Yes	The text assigned to this enumeration value.
SETPOINT_ALLOWED	Yes	Indicates if the point data field represented by this enumeration field can be set.

POINT_TYPE

Contains Point Type information.

Attribute ID	Filter	Description
POINT_TYPE_ID	Yes	The Point Type ID
DATA_TYPE	No	The numeric data type code for the point type.
DATA_LENGTH	No	The numeric data length for the point type.

PORT

Contains Port information.

Attribute ID	Filter	Description
PORT_ID	Yes	The Port ID.
PROTOCOL_ID	No	Identifier for the communication protocol used by the port.
DESCRIPTION	No	Description displayed for that port.

PROJECTS

Contains information on Remote Projects.

Attribute ID	Filter	Description
PROJECT_NAME	Yes	Project Name
USER_ID	No	The User ID to log into the project.
PASSWORD	No	Encrypted password for project login.
ENABLE	No	Indicates if the project is enabled.
EXCLUSIVE	No	Indicates if the project is exclusive.
CONCPOINTS	No	For an Enterprise Server, indicates if points are collected.
CONCALARMS	No	For an Enterprise Server, indicates if alarms are collected.
RESOURCE_ID	No	For an Enterprise Server, the remote project's resource name.
DEVICE_ID	No	For an Enterprise Server, the remote project's device name.

PROTOCOL

Contains Protocol information.

Attribute ID	Filter	Description
PROTOCOL_ID	Yes	Protocol ID

RESOURCE

Contains Resource information.

Attribute ID	Filter	Description
RESOURCE_ID	Yes	The Resource ID.
DESCRIPTION	No	Description of the resource.
RESOURCE_TYPE	No	The Resource Type: SYSTEM or RESOURCE.
ALARM_MGR_ID	No	The Alarm Manager process that receives alarms for this resource.

ROLE

Contains Role information.

Attribute ID	Filter	Description
ROLE_ID	Yes	The Role ID.

SSPC

Contains Statistical Process Control Information.

Attribute ID	Filter	Description
SPC_GROUP	Yes	A group, or subgroup, of SPC measurements.
SPC_FILE	No	SPC Configuration Document file name.

SYS_PARMS

Contains global parameter information for the system.

Attribute ID	Filter	Description
PARM_ID	Yes	System Parameter ID
PARM_VALUE	No	Value of the system parameter.

UAFSETS

Valid Attribute Set Identifier

Attribute ID	Filter	Description
SET_NAME	Yes	Set Name
DESCRIPTION	No	Description of the valid set.

USER

Contains User Information.

Attribute ID	Filter	Description
USER_ID	Yes	The User ID.
ROLE_ID	Yes	The users Role ID.
PASSWORD	No	The users encrypted password.
USER_NAME	No	The users name.
ENABLE	No	Indicates if the user account is enabled or disabled.

USER_FIELDS

Contains Field Information for Point Attribute Sets.

Attribute ID	Filter	Description
SET_NAME	Yes	Set Name.
FIELD_NAME	No	Field Name.
START_BIT	No	Start Bit.
FIELD_SIZE	No	Field Size.
READ_WRITE	No	Indicates if the field is read-only or read/write. 0 = Read only 2 = Read/Write
UPD_DEVCOMM	No	Write to DevComm - Data will be sent to the associated devcom when this attribute is set.
SAVE_WARM-DATA	No	Preserve value - Indicates that this field should be saved on project shutdown.

CimProjectData (object)

Overview	<p>The <code>CimProjectData</code> object provides the ability to search and return specific pieces of a project's configuration. The underlying APIs used by the <code>CimProjectData</code> object are the same as those used to browse point configuration on a remote project. In general, this object provides a convenient way to retrieve a set of attributes based on specified filter criteria. This object provides a read-only capability. To write configuration, please see the help file for the CIMPLICITY Configuration Object Model.</p>
Example (CimBasic)	<pre> Sub Main() ' This example retrieves all points beginning with A for Device MY_PLC ' in project MY_PROJECT and displays the point id and resource id of ' each matching item. Dim d as new CimProjectData d.Project = "MY_PROJECT" d.Entity = "POINT" d.Filters = "POINT_ID=A*,DEVICE_ID=MY_PLC" d.Attributes = "POINT_ID,RESOURCE_ID" Dim p as string Dim r as String top: if d.GetNext(p,r) = TRUE then MsgBox "Point Id = " & p & " Resource Id = " & r goto top End if End Sub </pre>
Example (.NET)	<pre> using System; using System.Collections.Generic; using Proficy.CIMPLICITY; public class CPD { public void Main() { try { CimProjectData cpd = new CimProjectData(); </pre>

```

cpd.Project = "MY_PROJECT";

cpd.Entity = "POINT";

cpd.Attributes = "POINT_ID,RESOURCE_ID";

cpd.Filters = "POINT_ID=PGM*";

String[] vals = new String[2]; // returned attributes matching the filters
Cimplicity.Trace("Get project points with IDs starting with \"PGM\"");

int count = 0;

while (cpd.Next(vals) == Cimplicity.COR_SUCCESS)
{
    Cimplicity.Trace("ID: " + vals[0] + ", Resource: " + vals[1]);

    count++;
}

Cimplicity.Trace("Finished getting project points.");
}
catch (Exception x)
{
    Cimplicity.Trace("Failure: " + x.Message);
}
}

```

CimProjectData.Project (property, read/write)

Syntax	CimProjectData.Project
Description	String. Get/set the project to browse data from. Must be specified when used from CimView. For use in the Event Manager, the project name should be empty to browse the local project.
Example	<pre> Dim d as new CimProjectData d.project = "MY_PROJECT" </pre>

CimProjectData.Reset (method)

Syntax	CimProjectData.Reset
Description	Resets the list so that a new set of search criteria, attributes, or project may be specified.
Example	<pre>d.reset</pre>

CimRemoveUnusedPoints (method)

Syntax	<code>CimRemoveUnusedPoints</code>
Description	Removes unused points that have been created as a result of Variable assignments.
Comments	The use of variables in expressions allows a user to assign points to animations at runtime. As the program makes various variable assigns and adds new points to CimView, it leaves other points in a state that no objects are using them. <code>CimRemoveUnusedPoints</code> can be used to remove these unused points from the screen, which reduces the number of updates, CimView receives from PTMAP, thus improving performance.
Example	<pre>Sub Cleanup CimRemoveUnusedPoints End Sub</pre>

DoQINTMath (function)

Syntax	<code>DoQINTMath param1,param2,param3,param4,param5,param6,param7</code>	
Description	To do the mathematics on a LONGLONG or QINT datatype in CIMPLICITY.	
	Param1	Double. High value of target 64-bit value
	Param2	Double. Low value of 64-bit target value
	Param3	Integer. Param3 is the input operator. Values represent the following.

		0	+
		1	-
		2	*
		3	/
		4	%
	Param4	Double. High value of source 1.	
	Param5	Double. High value of source 1.	
	Param6	Double . High value of source 2.	
	Param7	Double. Low value of Source 2.	
Example	<pre> Sub Main() Dim qlowSrc1 as Double Dim qhighSrc1 as Double Dim qlowSrc2 As Double Dim qhighSrc2 As Double Dim qtargetlow As Double Dim qtargethigh As Double Dim qstr as String DoUQINTMath qtargethigh,qtargetlow,0,qhighSrc1,qlowSrc1,qhighSrc2,qlowSrc2 - Addition qtargethigh,qtargetlow,1,qhighSrc1,qlowSrc1,qhighSrc2,qlowSrc2 - Subtraction End Sub </pre>		
See also	DoUQINTMath (on page 855) (function). Point.QuadValueAsString (on page 886) (property, read), Point.QuadValueAsString (on page 887) (property, write), Point.SetQuadIntValue (on page 897) (function).		

DoUQINTMath (function)

Syntax	DoUQINTMath param1,param2,param3,param4,param5,param6,param7
Description	To do the mathematics on a ULONGLONG or UQINT datatype in CIMPLICITY.

	Param1	Double. High value of target 64-bit value	
	Param2	Double. Low value of 64-bit target value	
	Param3	Integer. Param3 is the input operator. Values represent the following.	
		0	+
		1	-
		2	*
		3	/
		4	%
	Param4	Double. High value of source 1.	
	Param5	Double. High value of source 1.	
	Param6	Double . High value of source 2.	
	Param7	Double. Low value of Source 2.	
Example	<pre> Sub Main() Dim qlowSrc1 as Double Dim qhighSrc1 as Double Dim qlowSrc2 As Double Dim qhighSrc2 As Double Dim qtargetlow As Double Dim qtargethigh As Double Dim qstr as String DoQINTMath qtargethigh,qtargetlow,0,qhighSrc1,qlowSrc1,qhighSrc2,qlowSrc2 - Addition qtargethigh,qtargetlow,1,qhighSrc1,qlowSrc1,qhighSrc2,qlowSrc2 - Subtraction End Sub </pre>		
See also	DoQINTMath (on page 854) (function).		

GetCurTimeHR (function)

Syntax	<code>GetCurTimeHR</code>
Description	Date. To get the current time in MN_DD_YYYY_HH_MM_SS_TTTTTT format
Example	<pre> Sub Main() Dim timestr as String timestr = GetCurTimeHR MsgBox "Current time = " & timestr End Sub </pre>
See also	GetTimeComponentsHR (on page 864) (function).

GetKey (function)

Syntax	a\$ = GetKey (key\$, string\$)	
Description	To search for a keyword and returns its value. This is of use particularly from the Basic Control Engine to extract the EVENT and ACTION, which caused the script to run. An empty string is returned if the key is not found.	
Comments	Parameter	Description
	key\$	String . The keyword to search for.
	string\$	String . The string to search for the keyword. The format of this string is keyword followed by an equal sign and the value. A comma separates multiple keyword value combinations.
Example	<pre> Sub Main() event_id\$= GetKey("EVENT", command\$) action_id\$ = GetKey("ACTION", command\$) ' Name\$ will contain PETE after this statement. name\$ = GetKey("NAME", "NAME=PETE, LOCATION=ALBANY") End Sub </pre>	

GetMemoryInfoSymbolSpace (statement)

Syntax	GetMemoryInfoSymbolSpace used, free, total	
Description	This statement obtains information on the memory usage for storing the names of the symbols for public variables used in scripts at the module level.	
	Parameter	Description
	used	Long. The amount of memory in bytes that has been used for public variable space storage.
	free	Long. The amount of free space to hold new variable names.
	total	Long. The amount of memory available for public variable names.
Example	<pre> Public testPublicLong As Long Public testPublicString As String Private pv_test As Long Private pv_testString As String Type ExampleRect left As Integer top As Integer right As Integer bottom As Integer End Type Public dd As ExampleRect Sub OnMouseUp(x As Long, y As Long, flags As Long) Dim ssUsed As Long Dim ssFree As Long Dim ssMax As Long Dim psUsed As Long Dim psFree As Long Dim psMax As Long Dim SymUsed As Long Dim SymFree As Long Dim SymMax As Long Dim handlesUsed As Long </pre>	

```

Dim handlesFree As Long

Dim handlesMax As Long

Dim memFlags As Long

testPublicLong = 1200

pv_testString = 1200

testPublicString = "constant string to show usage of string space by constants"

pv_testString = "More data, more data"

GetMemoryInfoStringSpaceHandles handlesUsed, handlesFree, handlesMax

GetMemoryInfoStringSpace ssUsed, ssFree, ssMax, memFlags

GetMemoryInfoPublicSpace psUsed, psFree, psMax

GetMemoryInfoSymbolSpace SymUsed, SymFree, SymMax

MsgBox "The current memory information: " + Chr$(13)_
    + "Handles Used = " + Format$(handlesUsed) + Chr$(13)_
    + "Handles Free = " + Format$(handlesFree) + Chr$(13)_
    + "Handles Max = " + Format$(handlesMax) + Chr$(13)_
    + "String Space Used = " + Format$(ssUsed) + Chr$(13)_
    + "String Space Free = " + Format$(ssFree) + Chr$(13)_
    + "String Space Max = " + Format$(ssMax) + Chr$(13)_
    + "Public Space Used = " + Format$(psUsed) + Chr$(13)_
    + "Public Space Free = " + Format$(psFree) + Chr$(13)_
    + "Public Space Max = " + Format$(psMax) + Chr$(13)_
    + "Symbol Space Used = " + Format$(SymUsed) + Chr$(13)_
    + "Symbol Space Free = " + Format$(SymFree) + Chr$(13)_
    + "Symbol Space Max = " + Format$(SymMax)

End Sub

```

See [GetMemoryInfoStringSpaceHandles \(statement\) \(on page 859\)](#), [GetMemoryInfoStringSpace \(statement\) \(on page 861\)](#), [GetMemoryInfoPublicSpace \(statement\) \(on page 862\)](#)

GetMemoryInfoStringSpaceHandles (statement)

Syntax	GetMemoryInfoStringSpaceHandles used, free, total	
Description	This statement obtains information on the handle usage for string space.	
	Parameter	Description

	used	Long. The number of handles that have been used.
	free	Long. The number of handles that are free.
	total	Long. The total number of handles (32736).

Example	<pre> Option Explicit Sub OnMouseUp(x As Long, y As Long, flags As Long) Dim mymsg As String Dim used As Long, free As Long, total As Long, outFlags As Long Dim charCount Dim i Dim myarray(100) As String mymsg = "" mymsg = mymsg & Chr\$(13) & "---- BEFORE ----" GetMemoryInfoStringSpace used, free, total, outFlags mymsg = mymsg & Chr\$(13) & "SPACE used:" & used & ", free:" & free & ", total:" & total mymsg = mymsg & ", outFlags:" & outFlags GetMemoryInfoStringSpaceHandles used, free, total mymsg = mymsg & Chr\$(13) & "HANDLES used:" & used & ", free:" & free & ", total:" & total ' Use up some string space and handles charCount = 0 For i = LBound(myarray) To UBound(myarray) Step 1 myarray(i) = "ABCDEFGHJKLMNOPQRSTUVWXYZ " & i & " ABCDEFGHJKLMNOPQRSTUVWXYZ " charCount = charCount + Len(myarray(i)) Next i mymsg = mymsg & Chr\$(13) mymsg = mymsg & Chr\$(13) & "---- AFTER populating, elements:" & (UBound(myarray) - LBound(myarray)) _ & " char count:" & charCount & " ----" GetMemoryInfoStringSpace used, free, total, outFlags mymsg = mymsg & Chr\$(13) & "SPACE used:" & used & ", free:" & free & ", total:" & total mymsg = mymsg & ", outFlags:" & outFlags GetMemoryInfoStringSpaceHandles used, free, total mymsg = mymsg & Chr\$(13) & "HANDLES used:" & used & ", free:" & free & ", total:" & total MsgBox mymsg, vbOKOnly+vbInformation, "Memory Info" End Sub </pre>
----------------	---

See Also	GetMemoryInfoSymbolSpace (statement) (on page 857) , GetMemoryInfoStringSpace (statement) (on page 861) , GetMemoryInfoPublicSpace (statement) (on page 862)
----------	--

GetMemoryInfoStringSpace (statement)

Syntax	GetMemoryInfoStringSpace used, free, total[, outFlags]	
Description	This statement obtains information on the memory usage for string space.	
	Parameter	Description
	used	Long. The number of used bytes in the string space.
	free	Long. The number of free bytes in the string space.
	total	Long. The number of total bytes in the string space.
	outFlags	Long. The internal information about the string space. This parameter is unused.
Example	<pre> Option Explicit Sub OnMouseUp(x As Long, y As Long, flags As Long) Dim mymsg As String Dim used As Long, free As Long, total As Long, outFlags As Long Dim charCount Dim i Dim myarray(100) As String mymsg = "" mymsg = mymsg & Chr\$(13) & "---- BEFORE ----" GetMemoryInfoStringSpace used, free, total, outflags mymsg = mymsg & Chr\$(13) & "SPACE used:" & used & ", free:" & free & ", total:" & total mymsg = mymsg & ", outFlags:" & outFlags GetMemoryInfoStringSpaceHandles used, free, total mymsg = mymsg & Chr\$(13) & "HANDLES used:" & used & ", free:" & free & ", total:" & total </pre>	

<pre> ' Use up some string space and handles charCount = 0 For i = LBound(myarray) To UBound(myarray) Step 1 myarray(i) = "ABCDEFGHJKLMNOPQRSTUVWXYZ " & i & " ABCDEFGHIJKLMNOPQRSTUVWXYZ " charCount = charCount + Len(myarray(i)) Next i mymsg = mymsg & Chr\$(13) mymsg = mymsg & Chr\$(13) & "---- AFTER populating, elements:" & (UBound(myarray) - LBound(myarray)) _ & " char count:" & charCount & " ----" GetMemoryInfoStringSpace used, free, total, outFlags mymsg = mymsg & Chr\$(13) & "SPACE used:" & used & ", free:" & free & ", total:" & total mymsg = mymsg & ", outFlags:" & outFlags GetMemoryInfoStringSpaceHandles used, free, total mymsg = mymsg & Chr\$(13) & "HANDLES used:" & used & ", free:" & free & ", total:" & total MsgBox mymsg, vbOKOnly+vbInformation, "Memory Info" End Sub </pre>	
See	GetMemoryInfoSymbolSpace (statement) (on page 857) , GetMemoryInfoStringSpaceHandles (statement) (on page 859) , GetMemoryInfoPublicSpace (statement) (on page 862)
Note	The sum of the used and free parameter values will not be equal to the value of the total parameter. This is because of the overhead that is used to manage the allocated blocks.

GetMemoryInfoPublicSpace (statement)

Syntax	GetMemoryInfoPublicSpace used, free, total	
Description	This statement obtains information on the memory usage for storing the values for public variables used in scripts at the module level.	
	Parameter	Description
	used	Long. The amount of memory in bytes that has been used for public variable space storage.
	free	Long. The amount of free space to hold new variables.

	total	Long. The amount of memory available for public variables.
Ex- am- ple	<pre> Public testPublicLong As Long Public testPublicString As String Private pv_test As Long Private pv_testString As String Type ExampleRect left As Integer top As Integer right As Integer bottom As Integer End Type Public dd As ExampleRect Sub OnMouseUp(x As Long, y As Long, flags As Long) Dim ssUsed As Long Dim ssFree As Long Dim ssMax As Long Dim psUsed As Long Dim psFree As Long Dim psMax As Long Dim SymUsed As Long Dim SymFree As Long Dim SymMax As Long Dim handlesUsed As Long Dim handlesFree As Long Dim handlesMax As Long Dim memFlags As Long testPublicLong = 1200 pv_testString = 1200 testPublicString = "constant string to show usage of string space by constants" pv_testString = "More data, more data" GetMemoryInfoStringSpaceHandles handlesUsed, handlesFree, handlesMax GetMemoryInfoStringSpace ssUsed, ssFree, ssMax, memFlags GetMemoryInfoPublicSpace psUsed, psFree, psMax GetMemoryInfoSymbolSpace SymUsed, SymFree, SymMax MsgBox "The current memory information: " + Chr\$(13)_ + "Handles Used = " + Format\$(handlesUsed) + Chr\$(13)_ </pre>	

	<pre> + "Handles Free = " + Format\$(handlesFree) + Chr\$(13)_ + "Handles Max = " + Format\$(handlesMax) + Chr\$(13)_ + "String Space Used = " + Format\$(ssUsed) + Chr\$(13)_ + "String Space Free = " + Format\$(ssFree) + Chr\$(13)_ + "String Space Max = " + Format\$(ssMax) + Chr\$(13)_ + "Public Space Used = " + Format\$(psUsed) + Chr\$(13)_ + "Public Space Free = " + Format\$(psFree) + Chr\$(13)_ + "Public Space Max = " + Format\$(psMax) + Chr\$(13)_ + "Symbol Space Used = " + Format\$(SymUsed) + Chr\$(13)_ + "Symbol Space Free = " + Format\$(SymFree) + Chr\$(13)_ + "Symbol Space Max = " + Format\$(SymMax) End Sub </pre>
See	GetMemoryInfoStringSpaceHandles (statement) (on page 859) , GetMemoryInfoStringSpace
Also	(statement) (on page 861) , GetMemoryInfoPublicSpace (statement) (on page 862)

GetSystemWindowsDirectory (function)

Syntax	<code>d\$ = GetSystemWindowsDirectory</code>
Description	Returns the true Windows directory and not the per user Windows directory when running under Terminal Services.
Example	<pre> Sub Main() Direct\$ = GetSystemWindowsDirectory MsgBox "GetSystemWindowsDirectory = " & direct\$ End Sub </pre>

GetTimeComponentsHR (function)

Syntax	<code>GetTimeComponentsHR param1, param2, param 3 ...9</code>	
Description	Components of the time. Current time divided into time components of year, month, day, hour, min, sec and nanoseconds.	
	Param1	Double. High value of input time.
	Param2	Double. Low value of input time.
	Param3	Integer. Timecomponent.

	Param4	Integer. Timecomponent.
	Param5	Integer. Timecomponent.
	Param6	Integer. Timecomponent.
	Param7	Integer. Timecomponent.
	Param8	Integer. Timecomponent.
	Param9	Long. Nanosecond time component.
Example	<pre> Sub Main() Dim yy As Integer Dim mm As Integer Dim dd As Integer Dim hh As Integer Dim min As Integer Dim sec As Integer Dim nano As Long Dim localpoint As New Point Dim result As Boolean localpoint.id = "\\\$LOCAL\\$LOCAL.DATETIME_VARUPDATE" result = localpoint.GetQuadIntValue(qhigh,qlow) GetTimeComponentsHR qhigh,qlow,yy,mm,dd,hh,min,sec,nano End Sub </pre>	
See also	GetCurTimeHR (on page 856) (function), SetTimecomponentsHR (on page 918) (function)	

GetTSSessionId (function)

Syntax	<pre>id& = GetTSSessionId</pre>
Description	The Session ID of the Terminal Services client. This is 0 if running on the console or if Terminal Services is not running.
Example	<pre> Sub Main() myid& = GetTSSessionId </pre>

```

MsgBox "Terminal Services Session Id = " & myid&
End Sub

```

IsTerminalServices (function)

Syntax	<code>IsTerminalServices</code>
Description	Returns True if this computer is running Terminal Services.
Example	<pre> Sub Main() MsgBox "Terminal Services = " & IsTerminalServices End Sub </pre>

LogStatus (property, read/write)

Syntax	LogStatus Severity , Procedure\$, Message\$ [, error_code [, error_reference]]	
Description	To provide the programmer with the ability to log errors to the CIMPLICITY Status Log. To view the errors, use the CIMPLICITY Status Log Viewer.	
Comments	Parameter	Description
	Severity	<p>Integer. The severity of the error.</p> <ul style="list-style-type: none"> • CIM_SUCCESS - An Informational Error • CIM_WARNING - A warning message • CIM_FAILURE - A failure message
	Procedure\$	String. The name of the Basic Procedure which logged the error.
	Message\$	String. The error message to log.
	error_code	Long. (optional). A user-defined error code.
	error_reference	Long. (optional). A user-defined error reference. Used to distinguish the difference between two errors with the same error_code.

Example	<pre> Sub Main() on error goto error_handler Exit Sub error_handler : ' error\$, err, and erl are BASIC variables which contain the ' error text, error code and error line respectively. LogStatus CIM_FAILURE, "main()", error\$, err, erl Exit Sub End Sub </pre>
---------	--

Point.AlarmAck (property, read)

Syntax	Point.AlarmAck
Description	Boolean. When used in combination with the Point.OnAlarmAck method, a Boolean is returned indicating if the point's alarm is in an Acknowledged state.
Example	<pre> Sub Main() Dim x as new Point x.ID = "Some_point" x.OnAlarmAck top: x.GetNext Trace "Alarm Ack state is " & x.AlarmAck End Sub </pre>

Point.Cancel (method)

Syntax	Point.Cancel
Description	To cancel the currently active OnChange , OnAlarm , OnTimed or OnAlarmAck request.
Example	<pre> Sub Main() Dim t as new Point </pre>

	<pre> t.Id = "TIME" ' Read the next two values of the point t.OnChange for i = 1 to 2 t.GetNext next I ' Cancel the onchange request. t.Cancel ' Get the point value every three seconds t.OnTimed 3 for i = 1 to 2 t.GetNext next I End Sub </pre>
See Also	Point.OnChange (on page 884) (method), Point.OnTimed (on page 885) (method), Point.OnAlarm (on page 882) (method), Point.OnAlarmAck (on page 884) (method)

Point.ChangeApproval (property, write)

Syntax	Point.ChangeApproval = a
Description	To set the Change Approval information for a point. Important: Change Approval must be set to an object of type CimChangeApprovalData in order to perform set point operations on a point that requires change approval..
Example	<pre> Sub Main() Dim MyPoint As New Point Dim obj As New CimChangeApprovalData 'Init Point Set MyPoint.Id = "MYPOINT" 'Init CimChangeApprovalData with prompts Select Case MyPoint.ChangeApprovalInfo Case CP_CHANGEAPPROVALPERFORM obj.PerformerUserid = AskBox("Performer Userid") obj.PerformerPassword = AskPassword("Performer Password") Case CP_CHANGEAPPROVALPERFORMVERIFY obj.PerformerUserid = AskBox("Performer Userid") </pre>

	<pre> obj.PerformerPassword = AskPassword("Performer Password") obj.VerifierUserid = AskBox("Verifier Userid") obj.VerifierPassword = AskPassword("Verifier Password") Case CP_CHANGEAPPROVALNONE End Select 'Copy our CimChangeApprovalData into the Point's ChangeApproval Set MyPoint.ChangeApproval = obj 'Set the point MyPoint.SetValue = InputBox("Setpoint") End Sub </pre>
See also	CimChangeApprovalData (on page 821) (Object), AlarmUpdateCA (on page 814) (Method), PointChangeApprovalInfo (on page 869) (property, read).

Point.ChangeApprovalInfo (property, read)

Syntax	Point.ChangeApprovalInfo
Description	Integer. To determine the level of change approval that is required to perform an operation.
Example	<pre> Sub OnMouseUp(x As Long, y As Long, flags As Long) Dim MyPoint As New Point 'Init Point Set MyPoint.Id = "MYPPOINT" 'Check ChangeApprovalInfo to see what is required for approval Select Case MyPoint.ChangeApprovalInfo Case CP_CHANGEAPPROVALPERFORM MsgBox "This Point requires a Performer for approval!" Case CP_CHANGEAPPROVALPERFORMVERIFY MsgBox "This Point requires a Performer and Verifier for approval!" Case CP_CHANGEAPPROVALNONE MsgBox "This Point does not require ChangeApproval!" End Select End Sub </pre>

See also	AlarmUpdateCA (on page 814) (Method), CimChangeApprovalData (on page 821) (Object), Point.ChangeApproval (on page 868) (property, write)
----------	--

Point.DataType (property, read)

Syntax	Point.DataType	
Description	Integer. To return the numeric data type of the point.	
Comments	The following are the possible return values.	
	Return Value	Description
	CP_DIGITAL	A digital or Boolean value. Range True or False
	CP_STRING	A character string.
	CP_USHORT	An unsigned short (8-Bit) integer.
	CP_UINT	An unsigned (16-Bit) integer.
	CP_UDINT	An unsigned long (32-Bit) integer, returned as a double precision floating point number.
	CP_SHORT	A signed short (8-bit) integer.
	CP_INT	A signed (16-bit) integer.
	CP_DINT	A signed long (32-bit) integer.
	CP_REAL	A double precision floating point.
	CP_-BITSTRING	A bitstring. Can only be returned as a character string.
	CP_STRUCT	A structure point. Structure points are not currently supported.
Example	<pre> if MyPoint.DataType = CP_STRING then a\$ = MyPoint.Value else a% = MyPoint.Value end if </pre>	
See Also	Point.PointTypeId (on page 886) (property, read)	

Point.DisplayFormat (property, read)

Syntax	Point.DisplayFormat
Description	String. To return a string containing the configured display format for the point.

Point.DownloadPassword (property, read)

Syntax	Point.DownloadPassword
Description	Boolean. To determine if a download password is required to set the point.
Example	<pre>' Prompt the user for the download password if required to set ' the point. Sub Main() Dim p as new Point p.Id = "CP_UINT" p.Value = 10 if p.DownloadPassword then pass\$ = AskPassword("Download Password:") p.Set pass\$ else p.Set end if End Sub</pre>
See also	Point.SetPointPriv (on page 896) (property, read); Point.InUserView (on page 880) (property, read).

Point.Elements (property, read)

Syntax	Point.Elements
Description	Integer. To return the number of elements configured for the point. For array points this will be greater than 1, for non-array points the value will be 1.

Example	<pre> Sub Main() Dim MyPoint as new Point MyPoint.Id = "ARRAY_POINT" for x = 0 to MyPoint.Elements - 1 MyPoint.Value(x) = x next x MyPoint.Set End sub </pre>
----------------	--

Point.EnableAlarm (method)

Syntax	Point.EnableAlarm enable	
Description	To enable or disable alarming on the point. Can be used to temporarily disable alarming on a point.	
Comments	Parameter	Description
	Enable	Boolean. A value of TRUE enables alarming for the point and value of FALSE disables alarming for the point.
Example	<pre> Sub Main() Dim myPoint As New point myPoint.Id = "ALARM_POINT" ' Disable alarm for point. myPoint.EnableAlarm FALSE End Sub </pre>	

Point.Enabled (property, read)

Syntax	Point.Enabled
Description	Boolean. To determine if the point is enabled to be collected from the PLC.
	<pre> ' Return if the point is disabled. If MyPoint.Enabled = FALSE then Exit Sub end if </pre>

Point.EuLabel (property, read)

Syntax	Point.EuLabel
Description	String . To retrieve the Engineering Units Label for a point.
Example	<pre>a\$ = MyPoint.EuLabel</pre> <p>or</p> <pre>if MyPoint.EuLabel = "Litres" then ... end if</pre>

Point.Get (statement)

Syntax	Point.Get
Description	To get the current value of the point from the CIMPLICITY Point Manager and store it in the object. You may inspect the value through the Value and RawValue properties.
Example	<pre>Sub Main() Dim MyPoint as new Point MyPoint.Id = "\\PROJECT1\POINT1" MyPoint.Get MsgBox "The value is " & MyPoint.Value End Sub</pre>
See also	Point.Value (on page 906) (method), Point.OnChange (on page 884) (method), Point.OnTimed (on page 885) (method).

Point.GetArray (statement)

Syntax	Point.GetArray array [, startElement [, endElement [, fromElement]]]
--------	---

<p>De- scrip- tion</p>	<p>To retrieve an array point's values directly into a Basic array using Engineering Units Conversion if applicable. There are several rules to keep in mind:</p> <ul style="list-style-type: none"> • If the array is undimensioned, the array will be re-dimensioned to the same size as the point. • If the array is dimensioned smaller than the point, only that many elements will be copied into the array. • If the array is larger than the point, all elements of the point are copied, and the rest of the array is left as is. <p>If the startElement is specified, the function will start copying data into the array at this element and will continue until the end of the point is reached or the array is full whichever occurs first. If the endElement is specified, the function will stop copying data into the array after populating this element or when the end of the point is reached. If the fromElement is specified, the values copied into the array start at this element in the point array and continue as described above.</p> <p>Note: You must get the point value using the Get or GetNext method prior to using the GetArray method. The GetArray method does not retrieve the current value from the Point Manager. Instead, it retrieves the current value in the Point Object, which was generated during the last Get or GetNext . See the example below.</p>	
<p>Com- ments</p>	<p>Parameter</p>	<p>Description</p>
	<p>array</p>	<p>Array. A dimensioned or un-dimensioned Basic Array to which the point data will be copied.</p>
	<p>startElement</p>	<p>Integer. (optional) The first array element to which data will be copied.</p>
	<p>endElement</p>	<p>Integer. (optional) The last array element to which data will be copied.</p>
	<p>fromElement</p>	<p>Integer. (optional) The first point element from which data is to be copied.</p>
<p>Exam- ple</p>	<pre> Sub Main() Dim values() as integer Dim p as new Point ' Declare the point object p.Id = "ARRAY_POINT" ' Set the Id p.Get ' Get value from CIMPLICITY </pre>	

	<pre>p.GetArray values ' Copy the object into values End Sub</pre>
See Also	Point.SetArray (on page 894) (method); Point.GetRawArray (on page 877) (method); Point.HasEuConv (on page 879) (property, read); Point.Value (on page 906) (property, read/write); Point.RawValue (on page 891) (property, read/write).

Point.GetNext (function)

Syntax	Point.GetNext [(timeout)]
Description	Boolean. A function, to read the next value of a point with a specified timeout in milliseconds. Returns True if the point was read, False if it timed out.
Example	<pre>Sub Main() Dim MyPoint as new Point MyPoint.Id = "TIME" ' Set the Id MyPoint.OnChange ' Request the value on change MyPoint.GetNext ' The current value is returned immediately. if MyPoint.GetNext(1000) then ' Wait 1 second for the next value. MsgBox MyPoint.Value ' Display the value. Else MsgBox "Timeout" ' Point didn't change in one second. end if End Sub</pre>
See also	Point.OnChange (on page 884) (method); Point.OnTimed (on page 885) (method); Point.OnAlarm (on page 882) (method); Point.OnAlarmAck (on page 884) (method); Point.Cancel (on page 867) (method).

Point.GetNext (statement)

Syntax	Point.GetNext
Description	To wait for and get the next value of the point. This method returns when a point update is received for the point, based on a previously submitted OnChange , OnAlarm , OnTimed or OnAlarm .

	mAck call. If the point never changes, the call never returns. To wait with a timeout, see the GetNext (function.)
Ex-ample	<pre> ' Calculate the average of the next two point values. Sub Main() Dim MyPoint as new Point MyPoint.Id = "TANK_TEMPERATURE" ' Set the Id MyPoint.OnChange ' Request point onchange MyPoint.GetNext ' Retrieve the first value. x = MyPoint.Value ' Record the value. MyPoint.GetNext ' Wait for the next value. x1 = MyPoint.Value ' Record the value ave# = (x + x1)/ 2 ' Calculate the average MsgBox "The average was " & str\$(ave) End Sub </pre>
See Also	Point.OnChange (on page 884) (method); Point.OnAlarm (on page 882) (method); Point.OnTimed (on page 885) (method); Point.OnAlarmAck (on page 884) (method)..

Point.GetQuadIntValue (function)

Syntax	Point.GetQuadIntValue param1,param2	
Description	Will return the value of a 64-bit QINT or QUINT point in the form of two 32-bit double integers.	
	Param1	Double. High value.
	Param2	Double. Low value.
Example	<pre> Sub OnMouseUp(x As Long, y As Long, flags As Long) 'Declare variables Dim qhigh As Double Dim qlow As Double Dim result As Boolean Dim localpoint As New Point 'Initialize localpoint.id = "\\\$LOCAL\\$LOCAL.DATETIME_VARUPDATE" 'Gets the value of a QuadInt and places it in our two 32 bit Basic doubles </pre>	

	<pre> result = localpoint.GetQuadIntValue(qhigh,qlow) If result = True Then MsgBox qhigh MsgBox qlow Else MsgBox "Error!" End If End Sub </pre>
See also	Point.SetQuadIntValue (on page 897) (function)

Point.GetRawArray (statement)

Syn- tax	Point.GetRawArray array [, startElement [, endElement [, fromElement]]]	
De- scrip- tion	To retrieve an array points value directly into a Basic array bypassing Engineering Units Conversion.	
Com- ments	<p>There are several rules to keep in mind.</p> <ul style="list-style-type: none"> • If the array is undimensioned, the array will be re-dimensioned to the same size as the point. • If the array is dimensioned smaller than the point, only that many elements will be copied into the array. • If the array is larger than the point, all elements of the point are copied, and the rest of the array is left as is. <p>If the startElement is specified, the function will start copying data into the array at this element and will continue until the end of the point is reached or the array is full whichever occurs first. If the endElement is specified, the function will stop copying data into the array after populating this element or when the end of the point is reached. If the fromElement is specified, the values copied into the array start at this element in the point array and continue as described above.</p>	
	Parameter	Description
	array	Array . A dimensioned or un-dimensioned Basic Array to which the point data will be copied.

	startElement	Integer. (optional) The first array element to which data will be copied.
	endElement	Integer. (optional) The last array element to which data will be copied.
	fromElement	Integer. (optional) The first point element from which data is to be copied.
Example	<pre> Sub Main() Dim rawValues() as integer Dim p as new Point ' Declare the point object p.Id = "ARRAY_POINT" ' Set the Id p.Get ' Get value from SIMPLICITY p.GetRawArray rawValues ' Copy the object into values End Sub </pre>	
See Also	Point.GetArray (on page 873) (method); Point.SetRawArray (on page 897) (method); Point.HasEuConv (on page 879) (property/read); Point.Value (on page 906) (property, read/write); Point.RawValue (on page 891) (property, read/write).	

Point.GetTimeStampHR (statement)

Syntax	<code>Point.GetTimeStampHR param1,param2</code>	
Description	Date. To retrieve the Microsecond timestamp into 2 double 32-bit values. The timestamp indicates the time at which the point's value was read from the PLC.	
	Param1	Double. High value of the time
	Param2	Double. Low value of the time.
Example	<pre> Sub Main() Dim x as new Point Dim qhigh as Double Dim qlow as Double a\$ = InputBox\$("Enter a point id") x.Id = a\$ </pre>	

	<pre>x.GetTimeStampHR(qhigh,qlow) End Sub</pre>
See also	Point.QuadValueAsString (on page 886) (property, read), Point.QuadValueAsString (on page 887) (property, write), Point.SetQuadIntValue (on page 897) (function), Point.TimeStampHR (on page 878) (property, read), GetTimeComponentsHR (on page 864) (function), GetCurTimeHR (on page 856) (function).

Point.GetValue (property, read)

Syn-tax	Point.GetValue
De-scrip-tion	<p>To get a snapshot of the point value from the Point Manager and return it. This operation combines the Get Method and Value Property into a single command.</p> <p>If the point is unavailable (due to the device being down, remote server unavailable, etc.) an error will be generated if you attempt to access the value (since the value is unavailable.) See the Point.State property if you need to determine if the point is available or not.</p>
Ex-ample	<pre>Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the point id x = MyPoint.GetValue ' Read and return the value. End Sub</pre>

Point.HasEuConv (property, read)

Syn-tax	Point.HasEuConv
De-scrip-tion	Boolean. To determine if the point has Engineering Units conversion configured.
Ex-ample	<pre>Sub Main() Dim MyPoint as new Point MyPoint.Id = "DEVICE_POINT_1" if MyPoint.HasEuConv then MsgBox "Has Eu Conversion"</pre>

	<pre> else MsgBox "No Eu Conversion" end if End Sub </pre>
See also	Point.SetRawArray (on page 897) (method); Point.SetArray (on page 894) (method); Point.GetArray (on page 873) (method); Point.GetRawArray (on page 877) (method); Point.Value (on page 906) (property, read/write); Point.RawValue (on page 891) (property, read/write).

Point.Id (property, read/write)

Syntax	Point.Id	
Description	String. To get or set the object's CIMPLICITY Point ID. The function generates an error if the point is not configured or the remote server is not available.	
Comments	If an error is generated, one of the following error codes may be reported.	
	Err	Description
	CP_POINT_NOTFOUND	The Point ID specified is invalid and was not found.
Example	<pre> Sub Main() Dim MyPoint as new Point MyPoint.Id = "\\PROJECT1\POINT1" ' Set the id End Sub sub processPoint(MyPoint as Point) if MyPoint.Id = "GEF_DEMO_COS" then ' Compare the Id ... end if End Sub </pre>	

Point.InUserView (property, read)

Syntax	Point.InUserView
--------	-------------------------

De- scrip- tion	<p>Boolean. To determine if the point is in the user's view.</p> <ul style="list-style-type: none"> • If Resource Setpoint Security is checked in the Point Setup dialog box for the point's project and the point's resource is not in the user's view, then FALSE is returned. • If Level Setpoint Security is checked in the Point Setup dialog box and the point's level is greater than the level of the user's role, then FALSE is returned. • Otherwise, TRUE is returned. • If the point is not in the user's view, a run time error will be generated if you try to set it.
Ex- am- ple	<pre>Sub Main() Dim MyPoint as new Point MyPoint.Id = "TEST_POINT" if MyPoint.InUserView = TRUE MyPoint.SetValue = 10 else MsgBox "Point not in user view, setpoint not allowed" end if End Sub</pre>
See also	<p>Point.SetPointPriv (on page 896) (property, read); Point.DownloadPassword (on page 871) (property, read).</p>

Point.Length (property, read)

Syntax	Point.Length
Descrip- tion	Integer. To return the length in Bytes of the point value. This is valid only for character strings.
See also	Point.Elements (on page 871) (property, read)

Point (object)

Overview	The Point object provides an object-oriented interface to CIMPLICITY real-time point data. Through the object, you may set and read point values. Methods are supplied to receive the point value as it changes, periodically, or when the alarm state changes.	
Example	1	<code>Dim MyPoint as new Point</code> ' Creates a new empty point object
	2	<code>Dim ThisPoint as Point</code> ' Creates a pointer to a point object

	Set ThisPoint = MyPoint ' Now the two object are equal (BCE)	
Notes	In the above example, a point object is created two different ways.	
	1	Uses a new keyword; this is typically the method you will use. This constructs a point object, at which time you can set the ID of the point and use it.
	2	Creates a reference to a point and sets it to empty.
	<p>A runtime error will occur if you attempt to access methods of the object, since it is currently unassigned. You can assign the reference to a particular object by using the <code>SET</code> command. In general, you will use this with the PointGetNext function, which takes a list of point objects and returns the first one that changes.</p> <div style="border: 1px solid #ffc107; border-radius: 10px; padding: 10px; background-color: #fff3cd;"> <p>! Important: Point objects in <code>.NET</code> scripting must be explicitly disposed of by doing either of the following:</p> <ul style="list-style-type: none"> • Calling the <code>Point.Dispose()</code> method • Putting them inside the using block. </div> <p>Failing to do so will freeze the IDE (on page 252) when the script is finished running.</p>	

Point.OnAlarm (statement)

Syntax	Point.OnAlarm [cond1 [, cond2 [, cond3 [, cond4]]]]	
Description	To request the point's value when its alarm state changes. If no parameters are specified, the value will be returned whenever the alarm state changes. The four optional parameters can be used to restrict which alarm conditions will be reported to the application.	
Comments	Call GetNext to obtain the next value of the point. Only one of the OnChange , OnAlarm , OnTimed or OnAlarmAck requests may be active at a time. Optional Parameters	
	Value	Description
	CP_ALARM	Send the value whenever the point changes into or out of an Alarm (Hi or Low) state.

	CP_WARNING	Send the value whenever the point changes into or out of a Warning (Hi or Low) or Alarm (Hi or Low) state.
	CP_ALARM_HIGH	Send the value whenever the point changes into or out of an Alarm High state.
	CP_ALARM_LOW	Send the value whenever the point changes into or out of an Alarm Low state.
	CP_WARNING_HIGH	Send the value whenever the point changes into or out of a Warning High or Alarm High state.
	CP_WARNING_LOW	Send the value whenever the point changes into or out of a Warning Low or Alarm Low state.
Example	<pre> Sub Main() Dim MyPoint as new Point MyPoint.Id = "TANK_LEVEL" MyPoint.OnAlarm Top: MyPoint.GetNext if MyPoint.State = CP_ALARM_HIGH then MsgBox "Alarm High" elseif MyPoint.State = CP_ALARM_LOW then MsgBox "Alarm Low" elseif MyPoint.State = CP_WARNING_HIGH then MsgBox "Warning High" elseif MyPoint.State = CP_WARNING_LOW then MsgBox "Warning Low" elseif MyPoint.State = CP_UNAVAILABLE then MsgBox "Unavailable" else MsgBox "Normal" end if goto top End Sub </pre>	

See Also	Point.GetNext (on page 875) (method); Point.Cancel (on page 867) (method); Point.OnAlarmAck (on page 884) (method).
Notes	<p>The point value is sent when the point goes to warning or alarm state (based on the selected value), and then point value is sent again when the point goes back to normal state.</p> <p>Due to a current limitation, selecting ALARM_HIGH and WARNING_LOW, for example, will return the point for all alarm and warning states. In other words, the High and Low end up applying to both the Alarm and Warning.</p>

Point.OnAlarmAck (statement)

Syntax	Point.OnAlarmAck
Description	To receive the point's value when the alarm acknowledgment state changes.
	Only one of the OnChange , OnAlarm , OnTimed or OnAlarmAck requests may be active at a time.
See also	Point.GetNext (on page 875) (method); Point.Cancel (on page 867) (method); Point.OnAlarm (on page 882) (method).

Point.OnChange (statement)

Syntax	Point.OnChange
Description	To request the point's value on change. The next value of the point may be received by calling the GetNext method or function. The current value of the point is returned immediately. Any subsequent GetNext call will block until the point's value changes.
	Only one of the OnChange , OnAlarm , OnTimed or OnAlarmAck requests may be activate at a time.
Example	Read the point value on change forever.

	<pre> Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the Id MyPoint.OnChange ' Request the value on change top : MyPoint. GetNext ' Get the value Trace MyPoint.Value ' trace it to the output window goto top ' repeat forever End Sub </pre>
See also	Point.GetNext (on page 875) (method); Point.OnTimed (on page 885) (method); Point.Cancel (on page 867) (method).

Point.OnTimed (statement)

Syntax	Point.OnTimed time_period	
Description	To poll the points value periodically. A new value will be sent to the application every time_period seconds. The application should call GetNext to retrieve the next value.	
Comments	Unlike the OnChange method, you may miss values of the point if it changes in between your polls. Use the OnChange method to receive the point whenever it changes. OnTimed is useful if the point is rapidly changing and you are only interested in its value in a periodic manner. Only one of the OnChange , OnAlarm , OnTimed or OnAlarmAck requests may be active at a time.	
	Parameter	Description
	time_period	Integer . Time period in seconds to read the point.
Example	<pre> Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the point Id MyPoint.OnTimed 60 ' Request value every minute Top : MyPoint.GetNext ' Read the value Trace MyPoint.Value ' Put it out to the trace buffer goto top ' Repeat forever End Sub </pre>	

See	Point.GetNext (on page 875) (method); Point.OnChange (on page 884) (method); Point.Cancel (on page 867) (method).
Also	

Point.PointTypeId (property, read)

Syntax	Point.PointTypeId
Description	String. To retrieve the character based Point Type ID.
Example	<pre> Sub Main() Dim MyPoint as new Point MyPoint.Id = "CP_DIGITAL" if MyPoint.PointTypeId = "DIGITAL" then MsgBox "It is a digital point" else MsgBox "Point Type ID is : " & MyPoint.PointTypeId end If End Sub </pre>
See Also	Point.DataType (on page 870) (property, read)

Point.QuadValueAsString (property, read)

Syntax	<code>Point.QuadValueAsString</code>
Description	String. To return the string for the point values that are QINT,UQINT. Converts LONGLONG or ULONGLONG values of datatypes QINT or UQINT into strings and returns them.
Example	<pre> Sub Main() Dim p as new Point Dim val as String p.Id = "UQINT" val = p.QuadValueAsString MsgBox val End Sub </pre>

See also	Point.QuadValueAsString (on page 887) (property, write), write,) Point.SetQuadIntValue (on page 897) (function), Point.TimeStampHR (on page 878) (property, read).
----------	--

Point.QuadValueAsString (property, write)

Syntax	<code>Point.QuadValueAsString</code>
Description	Boolean. To take string of digits and convert them into 64-bit values and set the point values.
Example	<pre> Sub Main() Dim p as new Point Dim val as String p.Id = "UQINT" p.QuadValueAsString = "1234567899876543212" `Sets the value of the point that has type UQINT. End Sub </pre>
See also	Point.QuadValueAsString (on page 886) (property, read), Point.SetQuadIntValue (on page 897) (function), Point.TimeStampHR (on page 878) (property, read), GetTimeComponentsHR (on page 864) (function), GetCurTimeHR (on page 856) (function).

Point.Quality (property, read)

Syntax	Point.Quality
Description	Long . Return the 16-bit quality mask for the point.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get MsgBox cstr(p.Quality) End Sub </pre>

Point.QualityAlarmed (property, read)

Syntax	Point.QualityAlarmed
Description	Boolean. Returns TRUE if the point is in alarm, FALSE otherwise.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityAlarmed then MsgBox "Point is in alarm" End If End Sub </pre>

Point.QualityAlarms_Enabled (property, read)

Syntax	Point.QualityAlarms_Enabled
Description	Boolean. Returns TRUE if alarming for the point is enabled, FALSE otherwise.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityAlarms_Enabled then MsgBox "Alarming is enabled" End If End Sub </pre>

Point.QualityDisable_Write (property, read)

Syntax	Point.QualityDisable_Write
Description	Boolean. Returns TRUE if setpoints have been disabled for the point, FALSE otherwise.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityDisable_Write Then MsgBox "Writing disabled for point" End If End Sub </pre>

```

End If
End Sub

```

Point.QualityIs_Available (property, read)

Syntax	Point.QualityIs_Available
Description	Boolean. Returns TRUE if the points value is available, FALSE if the value is unavailable.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityIs_Available = FALSE then MsgBox "Point is not available" End If End Sub </pre>

Point.QualityIs_In_Range (property, read)

Syntax	Point.QualityIs_In_Range
Description	Boolean. Returns TRUE if the current value of the point is in range, FALSE if the point is out of range. When a point is out of range its value is unavailable.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityIs_In_Range = FALSE then MsgBox "Point is out of range" End If End Sub </pre>

Point.QualityLast_Upd_Man (property, read)

Syntax	Point.QualityLast_Upd_Man
Description	Boolean. Returns TRUE if the current value of the point came from a manual update rather than a device read.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityLast_Upd_Man then MsgBox "Last Update Manual" End If End Sub </pre>

Point.QualityManual_Mode (property, read)

Syntax	Point.QualityManual_Mode
Description	Boolean. Returns TRUE if the point has been placed into Manual Mode, otherwise FALSE.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityManual_Mode then PointSet "VALVE_1_STATE", "In Manual" Else PointSet "VALVE_1_STATE", "" End If End Sub </pre>

Point.QualityStale_Data (property, read)

Syntax	Point.QualityStale_Data
Description	Boolean. Returns TRUE if the value of the point is stale, otherwise FALSE. For more information on stale data, see QUALITY.STALE_DATA (Attribute) (<i>on page</i> 189).

Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get if p.QualityStale_Data = TRUE MsgBox "Value is stale" End If End Sub </pre>
----------------	---

Point.RawValue (property, read/write)

Syntax	Point.RawValue [(index)]	
Description	Same as Point.Value except bypasses Engineering Units conversion if configured for the point. Will return into any type subject to some restrictions. All numeric types may be returned into any other numeric type and into string types. String and BitString types can only be returned into string types. If the variable being returned into does not have a type, the variable will be changed to the appropriate type, based on the point type.	
Comments	<p>The option base determines if the first element of an array point will be zero or one. If you do not explicitly set the option base, all arrays in Basic start at 0. If you set it to 1, all arrays in Basic start at 1. See the example below.</p> <p><code>.RawValue</code> does not return the underlying numerical value for an enumerated point. If you want to obtain the underlying numerical value.</p> <ol style="list-style-type: none"> 1. Define a point with the <code>.ID</code> field set to <code><point_id>.\$RAW_VALUE</code>. 2. Reference the <code>.value</code> field of this point. 	
	Parameter	Description
	index	Integer. (optional) The array element to access. Range depends on the option base setting.
Example 1	<p>' Increment the points raw value by one.</p> <pre> Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the Id </pre>	

```

MyPoint.Get          ' Read the point

x = MyPoint.RawValue ' Return the raw value

MyPoint.RawValue = x + 1 ' Set the raw value

MyPoint.Set          ' Write the value.

End sub

' Find the maximum raw value in the array.

Option base 1          ' Arrays start at one.

Sub Main()

    Dim MyPoint as new Point ' Declare point object

    MyPoint.Id = "ARRAY_POINT" ' Set the Point Id

    MyPoint.Get          ' Get the value of the point

    max = MyPoint.RawValue(1) ' Get first value (option base = 1)

    for I = 2 to MyPoint.Elements ' Loop through all elements

        if MyPoint.RawValue(I) > max then max = MyPoint.RawValue(I)

    next I

End Sub

' Set all elements of the array to 10

option base 0          ' Arrays start at 0 (default)

Sub Main()

    Dim MyPoint as new Point ' Declare the object

    MyPoint.Id = "ARRAY_POINT" ' Set the Id

    ' Loop through all elements. Since arrays are set to start

    ' at 0, the index of the last element is one less than the

    ' count of the elements.

    for I = 0 to MyPoint.Elements - 1

        MyPoint.RawValue(I) = 10 ' Set the raw value

    next I

    ' Values are not written to CIMPPLICITY until this

    ' set is executed.

    MyPoint.Set          ' Write the point

End Sub

```

Example 2 'Access both the enumerated text and the underlying numerical 'value for a point.

```

Sub Main()

    Dim p1 As New point

    Dim p2 As New point

    'get the enumerated value

    p1.id = "ENUMERATEDPOINT"

```

	<pre> p1.get trace "enumerated text for " & p1.id & " is " & p1.value 'get the underlying numerical value p2.id = "ENUMERATEDPOINT.\$RAW_VALUE" p2.get 'yes, we really mean p1.id, with p2.value!!! trace "underlying numeric value for " & p1.id & " Is " & p2.value End Sub </pre>
Note	Point.Value (on page 906) (property, read/write)

Point.ReadOnly (property, read)

Syntax	Point.ReadOnly
Description	Boolean. To determine if the point is read only.
Example	<pre> Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the Id if MyPoint.ReadOnly then ' Is the point read-only? MsgBox "Point cannot be set, point is read-only" else MyPoint.SetValue = 10 ' Set the value and write to CIMPLICITY. end if End Sub </pre>

Point.Set (statement)

Syntax	Point.Set [downloadPassword]
Description	To write the point's value out to the CIMPLICITY project. An optional download password can be supplied.
Comments	The values set into the Point using the <code>Value</code> , <code>RawValue</code> , <code>SetArray</code> and <code>SetRawArray</code> methods are not written out to the CIMPLICITY project until they are committed with a <code>Set</code> or <code>SetNoAudit</code> (on page 896) statement.

	Parameter	Description
	downloadPassword	String. (optional) The download password for the project.
Example	<pre> Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the Id MyPoint.Value = 10 ' Set the value MyPoint.Set ' Write the value out to SIMPLICITY End Sub </pre>	
See Also	Point.SetValue (on page 899) (property, write), and Point.SetNoAudit (on page 896) (statement)	

Point.SetArray (statement)

Syntax	Point.SetArray array [, startElement [, endElement [, fromElement]]]	
Description	To set an array point's values directly from a Basic array.	
Comments	<p>There are several rules to keep in mind:</p> <ul style="list-style-type: none"> • If the array is dimensioned smaller than the point, only that many elements will be copied into the point. • If the array is larger than the point, all elements of the array are copied, and the rest of the array is ignored. <p>If the startElement is specified, the function will start copying data from the array at this element and will continue until the end of the array is reached or the point is full whichever occurs first. If the endElement is specified, the function will stop copying data from the array after copying this element or when the point is full. If the fromElement is specified, the values copied from the array start at this element in the point array and continue as described above.</p>	
	Parameter	Description
	array	Array. A dimensioned or undimensioned Basic Array from which the point data will be copied.

	startElement	Integer. (optional) The first array element from which data will be copied.
	endElement	Integer. (optional) The last array element from which data will be copied.
	fromElement	Integer. (optional) The first point element to which data is to be copied.
Example	<pre> ' Read an array point, sort the elements by value and write them ' out to CIMPLICITY sorted. Sub Main() Dim x() as integer 'Declare the value array Dim MyPoint as new Point 'Declare the point object MyPoint.id = "POINTNAME" 'Assign point to script MyPoint.Get 'Get the point value MyPoint.GetArray x 'Transfer point element into array ArraySort x 'Sort the array MyPoint.SetArray x 'Transfer to array into the point MyPoint.Set 'Transfer the sorted data to CIMPLICITY. End Sub </pre>	
See Also	Point.SetRawArray (on page 897) (method); Point.Value (on page 906) (property, read/write), Point.GetArray (on page 873) (method); Point.Set (on page 893) (method).	
Note	The SetArray method only updates the internal value of the point object. The Set method must be executed to write the value out to the CIMPLICITY project.	

Point.SetElement (statement)

Syntax	Point.SetElement index, [download password]	
Description	To write a single element of the point to the Point Manager.	
Comments	Parameter	Description
	Index	Integer. The index of the element to write.
	download password	String. (optional) download password
Example	<pre> 'Set only the third element of an array Sub Main() </pre>	


```

Dim MyPoint As New Point      'Declare the point object

MyPoint.Id = "INT_ARRAY"

MyPoint.Value(3) = 10        'Assign the value of the third element

MyPoint.SetElement 3        'Write only the third element

End Sub

```

Point.SetNoAudit (statement)

Syn- tax	Point.SetNoAudit [downloadPassword]	
De- scrip- tion	To write the point's value to the CIMPLICITY project. Setpoint audit trail events will not be triggered when using this method. An optional download password can be supplied.	
Com- ments	The values set into the Point using the <code>Value</code> , <code>RawValue</code> , <code>SetArray</code> and <code>SetRawArray</code> methods are not written out to the CIMPLICITY project until they are committed with a Set (on page 893) or <code>SetNoAudit</code> statement.	
	Parameter	Description
	downloadPassword	String. (optional) The download password for the project.
Exam- ple	<pre> Sub Main() Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "TANK_LEVEL" ' Set the Id MyPoint.Value = 10 ' Set the value MyPoint.SetNoAudit ' Write the value out to CIMPLICITY End Sub </pre>	
See Also	Point.Set (on page 893) (statement)	

Point.SetpointPriv (property, read)

Syntax	<code>Point.SetpointPriv</code>
Description	Boolean. To determine if the user accessing the point has Setpoint privilege.
Example	<pre> Sub Main() Dim MyPoint as new Point </pre>

	<pre> MyPoint.Id = "TANK_LEVEL" if MyPoint.SetpointPriv = FALSE then MsgBox "You do not have the setpoint privilege" else MyPoint.SetValue = InputBox\$("Setpoint Value:") end if End Sub </pre>
See also	Point.DownloadPassword (on page 871) (property, read); Point.InUserView (on page 880) (property, read).

Point.SetQuadIntValue (function)

Syntax	<pre>Point.SetQuadIntValue(qhigh, qlow)</pre>
Description	<p>To set the point's value in a CIMPLICITY project. This operation combines the Value and Set operations into one command. The <code>SetQuadIntValue</code> function takes two double values to set the value of any 64 bit data type QINT or UQINT.</p>
Example	<pre> ' To set the value of any point with data type QINT or UQINT ' follow the example below. Sub Main() Dim qstr As String Dim qhigh as Double Dim qlow as Double Dim MyPoint as new Point 'Declare the point object MyPoint.Id = "QINT" 'Set the Id qstr = "1000000899876543212" QINTFromString qstr,qhigh,qlow SetQuadIntValue (qhigh,qlow) End Sub </pre>
See also	Point.QuadValueAsString (on page 886) (property, read), Point.QuadValueAsString (on page 887) (property, write), Point.SetQuadIntValue (on page 897) (function), Point.TimeStampHR (on page 878) (property, read); Point.GetQuadIntValue (on page 876) (function)

Point.SetRawArray (statement)

Syn-tax	Point.SetRawArray array [, startElement [, endElement [, fromElement]]]	
De-scrip-tion	To set an array point's values directly from a Basic array, bypassing Engineering Units Conversion.	
Com-ments	<p>There are several rules to keep in mind:</p> <ul style="list-style-type: none"> • If the array is dimensioned smaller than the point, only that many elements will be copied into the point. • If the array is larger than the point, all elements of the point are set. <p>If the startElement is specified, the function will start copying data from the array at this element and will continue until the end of the array is reached or the point is full whichever occurs first. If the endElement is specified, the function will stop copying data from the array after copying this element or when the point is full. If the fromElement is specified, the values copied from the array start at this element in the point array and continue as described above.</p>	
	Parameter	Description
	array	Array. A dimensioned or undimensioned Basic Array from which the point data will be copied.
	startElement	Integer. (optional) The first array element from which data will be copied.
	endElement	Integer. (optional) The last array element from which data will be copied.
	fromElement	Integer. (optional) The first point element to which data is to be copied.
Exam-ple	<pre>' Copy the log value of one array point to another array point. Sub Main() Dim source as new Point ' Declare source point Dim dest as new Point ' Declare destination point Dim x() as double ' Declare array source.Id = "INPUT" ' Set the ID of the source point source.Get ' Get the value of the source point dest.Id = "OUTPUT" ' Set the ID of the destination point source.GetRawArray x ' Transfer value to array</pre>	

	<pre> ' Loop through array point, taking logarithm. for I = 0 to source.Elements - 1 x(I) = log(x(I)) next I dest.SetRawArray x ' Transfer value into destination object dest.Set ' Set the value to CIMPLICITY End Sub </pre>
See Also	Point.SetArray (on page 894) (method); Point.RawValue (on page 891) (property, read/write); Point.GetRawArray (on page 877) (method).
Note	<p>The <code>SetRawArray</code> method only updates the internal value of the point object. The Set method must be executed to write the value out to the CIMPLICITY project.</p>

Point.SetValue (property, write)

Syntax	Point.SetValue = a
Description	<p>To set the point's value in a CIMPLICITY project. This operation combines the Value and Set operations into one command. The SetValue method uses Engineering Units Conversion and cannot be used to set elements of an array point.</p>
Example	<pre> ' Ramp tank level from 0 to 100 in steps of five, with a delay ' on 100ms between each set. Sub Main() Dim MyPoint as new Point 'Declare the point object MyPoint.Id = "TANK_LEVEL" 'Set the Id for I = 0 to 100 step 5 'Loop in steps of 5 MyPoint.SetValue = I 'Set and write value to CIMPLICITY Sleep 100 'Sleep 100ms next I 'Loop End Sub </pre>

Point.State (property, read)

Syntax	Point.State
Description	Integer. To return the state of the point's value.

Comments	Any of the following states may be returned.	
	State	Description
	CP_NORMAL	Point is in Normal State
	CP_ALARM_HIGH	Point is in Alarm High State.
	CP_ALARM_LOW	Point is in Alarm Low State.
	CP_WARNING_HIGH	Point is in Warning High State.
	CP_WARNING_LOW	Point is in Warning Low State.
	CP_ALARM	Point is in Alarm State.
	CP_WARNING	Point is in Warning State.
	CP_AVAILABLE	Point has gone from Unavailable to Available.
	CP_UNAVAILABLE	Point is Unavailable
Example	<pre> ' Increment the point value by one, if the point is unavailable, ' set it to 0. Sub Main() Dim MyPoint as new Point MyPoint.Id = "TANK_LEVEL" MyPoint.Get if MyPoint.State = CP_UNAVAILABLE then MyPoint.SetValue = 0 else MyPoint.SetValue = MyPoint.Value + 1 end if End Sub </pre>	
See Also	Point.Get (on page 873) (method); Point.GetNext (on page 875) (method)	

Point (subject)

Overview	The values of CIMPLICITY points can be used in a variety of ways by a script. You can use scripts that act on point values to define reactions to changing conditions in your process.
----------	--

	<p>Points are manipulated by the PointSet statement and PointGet function or the point object. In general, PointSet and PointGet are useful if you require the value of the point or wish to set the point. The point object extends your capabilities by allowing you to receive point values as they change, access array points, provide more information about the point's configuration; and improve performance when repeatedly setting a point.</p>
Security	<p>The CIMPLICITY extensions to Basic provide the same security which all your CIMPLICITY applications use; Set Point Security, Set Point Privilege, Download Password and Set Point Audit trail. In order to discuss security, first we will need to understand when security is imposed on your access to points. There are two categories of processes running on your CIMPLICITY Server; User Applications and Resident Processes. User Applications are applications run by the user, that usually provide a user interface. Examples of such programs are CimView, CimEdit, Alarm Viewer and Program Editor. In order for the application to access a point on the local CIMPLICITY project or a remote CIMPLICITY project, a user login is required. The CIMPLICITY privileges defined for your User ID define your capabilities. Resident Processes are processes that are started as part of your CIMPLICITY project. Examples of resident processes are the Database Logger, Point Manager and scripts automatically run by the Basic Control Engine. Since a resident process is a trusted part of your system, a resident process is not required to obtain a login in order to access points in their project. If the resident process wishes to access a point on a remote system, a remote project must be configured to supply the resident process with the User ID and Password with which to log in to the remote system.</p>
Performance	<p>The CIMPLICITY extensions to Basic provide a high performance mechanism to interact with your Point Database. However, there are several considerations to keep in mind when designing your application to obtain the highest performance possible. First, is the Set Point Audit Trail. For each CIMPLICITY role, you may configure whether or not the user will generate an audit trail for each setpoint. The audit trail is composed of a \$DOWNLOAD event containing information on who set the point. This information is sent to your event log and can provide a detailed audit trail of who and what was set. However, the audit trail imposes significant overhead (20 times slower), since the record is logged to the database for each setpoint. This is particularly noticeable when running setpoints in a loop in the Program Editor. However, when the script is run from the Basic Control Engine, a \$DOWNLOAD event will not be generated since a resident process is trusted. If you do not require an audit trail it is recommended that you disable it through role configuration (this is the default).</p>
	<p>Second, is the difference between a PointSet statement and using the Point Object. With a Point Object, you create the object once and initialize its point information once (data type, elements, etc.). Subsequent operations on the Point are very fast, since the point characteris-</p>

	<p>tics are contained in the object. Conversely, PointSet and PointRead must fetch the point information on each execution (in benchmark testing this is 2 times slower.) Consider the following example :</p>
	<pre> ' Example One sub slow_set() for I = 0 to 100 PointSet "MY_POINT", I next I End Sub ' Example two sub fast_set Dim MyPoint as new Point MyPoint.Id = "MY_POINT" for I = 0 to 100 MyPoint.SetValue = I next I End Sub </pre>
	<p>The subroutine fast_set ramps the point ten times faster than the slow_set routine. While the second example at first may appear more complex, you will find that the object interface provides much more flexibility. As a rule, use PointGet and PointSet when you need to read or set the point's value once within your script.</p>
Polling	<p>CIMPLICITY provides a high performance Point Interface. As a result, improperly written applications can degrade the overall performance of a system. One common issue is polling a point to wait for it to change. Consider the following example. Incorrect Code</p> <pre> Poll: If PointGet("POLL_POINT") = 0 then Sleep 100 Goto poll End If </pre> <p>The sleep statement causes a 100ms delay between polls. However many extra polls are still being performed. Correct and Most Efficient Code</p> <pre> Dim p as new point p.Id = "POLL_POINT" p.Onchange Poll: </pre>

	<pre> Wait_for p.GetNext if p.Value=0 then goto wait_for </pre>
	<p>In this example, the script requests the value of the point as it changes. When the point changes, the GetNext statement returns. When the point is not changing the script is waiting and using no system resources.</p>
Error Handling	<p>Basic provides a flexible error handling capability with the On Error command. The CIMPLICITY extensions to Basic are designed to use the built in error handling capability. When an error occurs while executing your CIMPLICITY command, a Basic Run Time error is generated. There are many ways you can implement error handling. Among these are :</p> <ul style="list-style-type: none"> • No error handling. When an error occurs, the script's execution halts and the error is reported (in the Program Editor, this is via a Message Box, and in the control engine by logging an error message to the status log). • Error Handler. When an error occurs, the script's execution moves to the defined error handler. Within the error handler, the user can report the error or try to recover. • In line error checking. When an error occurs, the script's execution continues on the next program statement. The user can check the err variable to determine if an error occurred. <p>In the fast_set example above a run time error could be generated on the setting of the ID or the setting of the value. Since the routine provides no error handling, when an error occurs, the routine exits and returns to the calling routine. If no error handler is found as the program returns up the call stack, a default error handler reports the run-time error. If you run the script from the Program Editor, a dialog box opens, and if it is run from the Basic Control Engine, a Status Log message is created.</p>
	<p>Consider the two examples below:</p>
	<pre> Sub inline_errorcheck() ' When an error occurs continue execution at the next statement on error resume next PointSet "BAD_POINT", 10 ' Did an error occur? If err <> 0 then ' clear the error err = 0 End If End Sub </pre>

	<pre> End if PointSet "BAD_POINT1", 10 if err <> 0 then err = 0 Exit Sub end if End Sub sub outline_errorcheck() ' When an error occurs goto the error handler on error goto error_handler PointSet "BAD_POINT", 10 PointSet "BAD_POINT1", 10 Exit Sub error_handler: MsgBox "Error" Exit Sub End Sub </pre>
	<p>You can choose how to handle or not handle error conditions.</p>

Point.TimeStamp (property, read)

Syntax	Point.TimeStamp
De- scrip- tion	Date. To retrieve the timestamp into a Basic Date Object. The timestamp indicates the time at which the point's value was read from the PLC.
Exam- ple	<pre> Sub Main() Dim x as new Point a\$ = InputBox\$("Enter a point id") x.Id = a\$ x.OnChange top : x.GetNext Trace str\$(x.TimeStamp) & " " & x.Value goto top End Sub </pre>

See also	Point.Get (on page 873) (method); Point.GetNext (on page 875) (method).
----------	---

Point.TimeStampHR (property, read)

Syntax	<code>Point.TimeStampHR</code>
Description	Date. To retrieve the Microsecond timestamp into a string object. The timestamp indicates the time at which the point's value was read from the PLC.
Example	<pre> Sub Main() Dim x as new Point a\$ = InputBox\$("Enter a point id") x.Id = a\$ x.OnChange top : x.GetNext Trace str\$(x.TimeStampHR) & " " & x.Value goto top End Sub </pre>
See also	Point.Get (method); Point.GetNext (method), Point.GetTimeStampHR (method).

Point.UserFlags (property, read)

Syntax	Point.UserFlags
Description	Long. Returns the value of the 16-bit user defined flags for the point.
Example	<pre> Sub Main() Dim p as new Point p.Id = "VALVE_1" p.Get MsgBox cstr(p.UserFlags) End Sub </pre>

Point.Value (property, read/write)

Syntax	Point.Value [(index)]
Description	To retrieve or set the value in the point object. The optional index may be supplied to access values of an array point. The first element of the array is at the zero index. The value property uses Engineering Units conversion if supplied by the point. To bypass Engineering Units conversion, use the RawValue property.
	Automatic conversion will be performed between data types as needed. The only exceptions are String and BitString points, which can only be assigned from Strings.
Example	<pre>' This subroutine show automatic type conversion Sub Main() Dim MyPoint as new Point 'Declare the point object MyPoint.Id = "INTEGER_POINT" 'Set the Id, Point Type is INTEGER ' The string value of "10" is automatically converted to a integer ' value of 10 and place in point object. MyPoint.Value = "10" MyPoint.Set ' Write the point ' The floating point value of 10.01 is truncated to 10 and place ' in the point MyPoint.Value = 10.01 MyPoint.Set ' Write the point End Sub</pre>
See also	Point.RawValue (on page 891) (property, read/write); Point.GetArray (on page 873) (method); Point.GetRawArray (on page 877) (method).
Notes	<ul style="list-style-type: none"> • To retrieve the point value, the Point.Get method must be invoked first. Once the value has been read, it can be accessed many times without having to retrieve it from the Point Manager on each reference. If the point hasn't been read, an exception is generated. • When setting a value, the value is not written to the device until the Set method is invoked.

PointGet (function)

Syntax	PointGet (pointId\$)
--------	-------------------------------

Description	To read a particular point and return the value.	
Comments	Parameter	Description
	pointId\$	String . The Point ID to get the value from.
Example	<pre>' Prompt user for point id, get the point value and display ' it into a message box. Sub Main() MsgBox "Value is " & PointGet(InputBox\$("Enter Point Id")) End Sub</pre>	
See Also	PointGetMultiple (on page 908) (function)	

**Important:**

For CIMPLICITY Machine Edition's array point names

Enclose CIMPLICITY Machine Edition array point names (that are passed through CIMPLICITY Plant Edition Basic) in the the ASCII encoding for single quotes `Chr$(39)`.

The reason is as follows:

CIMPLICITY Machine Edition returns array points as single values using the form `name[index]`.

When a CIMPLICITY Machine Edition's array point name:

- Is not enclosed in `Chr$(39)`, BASIC will parse this out as a reference to an array element. You will receive an error indicating a bad point name.
- Is enclosed in `Chr$(39)` the point will not be parsed in the PointSet and PointGet BASIC procedures. The name will be passed straight through to Machine Edition.

**Note:**

You cannot directly put a single quote (') on an argument line because the single quote in Basic denotes that the remainder of the line is a comment.

Examples

- `val = PointGet("MyPointName")`

Result

`PointGet` receives `. MyPointName`

```
• val = PointGet(Chr$(39) & "MyPointName[10]" & Chr$(39))
```

Result

`PointGet` receives `'MyPointName[10]'`

PointGetMultiple (function)

Syntax	<code>PointGetMultiple point1[,point2[,point3...]]</code>	
Description	Request data from up to 30 points in a single snapshot request. If the function fails, an error is generated.	
Comments (Cim-Basic)	If you need to get data from several points, use this function rather than issuing a single PointGet command for each point. For the example below, it is six times more efficient to use PointGetMultiple , since the data is retrieved from the Point Manager in a single request, rather than six separate PointGet requests.	
	Parameter	Description
	<code>pointn</code>	Point objects for which data is going to be requested. Up to 30 may be specified as function parameters.
Example (Cim-Basic)	<pre>Sub Main() Dim x As New Point: x.Id = "R1" Dim x1 As New Point: x1.Id = "R2" Dim x2 As New Point: x2.Id = "R3" Dim x3 As New Point: x3.Id = "R4" Dim x4 As New Point: x4.Id = "R5" Dim x5 As New Point: x5.Id = "R6" PointGetMultiple x,x1,x2,x3,x4,x5 End Sub</pre>	
Comments (.NET)	<code>PointGetMultiple</code> has been ported to <code>.NET</code> as follows:	

```
void Cimplicity.PointGetMultiple(Point[] points);
```

`PointGetMultiple` takes an array of `Point` objects, which is different from `CimBasic` where `CimBasic` functions take variable arguments with each being a `Point` object. Otherwise `.NET` and `CimBasic` behavior is the same for this function.

**Example
(.NET)**

```
using System;
using System.Collections.Generic;
using Proficy.CIMPLICITY;

public class PGM
{
    public void Main()
    {
        Point[] array = new Point[3];

        using (Point one = new Point(), two = new Point(), three = new Point())
        {
            one.Id = "PGM_01";
            one.OnChange();

            two.Id = "PGM_02";
            two.OnChange();

            three.Id = "PGM_03";
            three.OnChange();

            array[0] = one;
            array[1] = two;
            array[2] = three;

            try
            {
                Cimplicity.PointGetMultiple(array);

                foreach (Point p in array)
                {
                    Cimplicity.Trace(p.Id + " -> " + p.Value.ToString());
                }
            }
        }
    }
}
```

	<pre> } catch (Exception x) { Cimplicity.Trace("Failure: " + x.Message); } } } } </pre>
See	PointGet (on page 873) (method)
Also	

PointGetNext (function)

Syntax 1	<code>PointGetNext(timeOutMs, point1 [... [, point16])</code>
Syntax 2	<code>PointGetNext(timeOutMs, PointArray)</code>
De- scrip- tion	To return the next point value from a list of points with a timeout.
Com- ments (Cim- Basic)	<p>Timeout values (milliseconds) can be as follows:</p> <ul style="list-style-type: none"> - 1 0 Positive Integer
	Point1 is a point object with an outstanding request. Up to 16 points can be specified on the function call. Alternatively, the user may pass an array of point objects. The function returns the object whose value changed or empty. Parameter: timeOutMs, pointn, PointArray.
Exam- ple	<pre> ' Trace the values of 2 point as they change or trace timeout if neither ' point change in 1 second. Sub Main() </pre>

(Cim-Basic)	<pre> Dim Point1 as new Point ' Declare Point Object Dim Point2 as new Point ' Declare Point Object Point1.Id = "TANK_LEVEL" ' Set the Id Point2.Id = "TANK_TEMP" ' Set the Id Point1.OnChange ' Register OnChange request Point2.OnChange ' Register OnChange request Dim Result as Point ' Declare result pointer Top : ' Set result equal to result of waiting on Point1 and Point2 ' to change for 1 second Set Result = PointGetNext(1000, Point1, Point2) if Result is Nothing then ' Nothing is returned if timeout Trace "TimeOut" Else ' Otherwise Result is Point1 or Point2 depending on which one ' changed last. Trace Result.Id & " " & str\$(Result.TimeStamp) & Result.Value end if goto top End Sub </pre>
Comments (.NET)	<p><code>PointGetNext</code> has been ported to .NET as follows:</p>
	<pre> Point Cimplicity.PointGetNext(int TimeOutMs, Point[] points); </pre>
	<p><code>PointGetNext</code> takes an array of Point objects, which is different from CimBasic where CimBasic functions take variable arguments with each being a Point object. Otherwise .NET and CimBasic behavior is the same for this function.</p>
Example (.NET)	<pre> using System; using System.Collections.Generic; using Proficy.CIMPLICITY; public class PGN { public void Main() { Point[] array = new Point[3]; </pre>


```
using (Point one = new Point(), two = new Point(), three = new Point())
{
    one.Id = "PGN_1";
    one.OnChange();

    two.Id = "PGN_2";
    two.OnChange();

    three.Id = "PGN_3";
    three.OnChange();

    array[0] = one;
    array[1] = two;
    array[2] = three;

    try
    {
        Point result;

        do
        {
            result = Cimplicity.PointGetNext(30000, array);

            if (result != null)
            {
                Cimplicity.Trace("Point that changed is " + result.Id);
            }
        } while (result != null);
    }
    catch (Exception x)
    {
        Cimplicity.Trace("Failure: " + x.Message);
    }
    finally
    {
        Cimplicity.Trace("No more changes after 30 seconds");
    }
}
```

```

    }
  }
}
}

```

PointSet (statement)

Syntax	PointSet pointId\$, value	
Description	To set a point's value.	
Comments	Parameter	Description
	pointId\$	String . The point ID to set.
	value	Value to set it to.
Example	<pre> Sub Main() PointSet InputBox\$("Point Id:"), InputBox\$("Value:") End Sub </pre>	



Important:

For CIMPLICITY Machine Edition's array point names

Enclose CIMPLICITY Machine Edition array point names (that are passed through CIMPLICITY Plant Edition Basic) in the the ASCII encoding for single quotes `Chr$(39)`.

The reason is as follows:

CIMPLICITY Machine Edition returns array points as single values using the form `name[index]`.

When a CIMPLICITY Machine Edition's array point name:

- Is not enclosed in `Chr$(39)`, BASIC will parse this out as a reference to an array element. You will receive an error indicating a bad point name.
- Is enclosed in `Chr$(39)` the point will not be parsed in the PointSet and PointGet BASIC procedures. The name will be passed straight through to Machine Edition.

**Note:**

You cannot directly put a single quote (') on an argument line because the single quote in Basic denotes that the remainder of the line is a comment.

Examples

- `PointSet "MyPointName", val`

Result

`PointSet` sets `MyPointName` to the value of `val`.

- `PointSet Chr$(39) & "MEArrayPointName[10]" & Chr$(39), val`

Result

`PointSet` sets the element with the index 10 of the Machine Edition array point `MEArrayPointName` to the value of `val`.

Important: This syntax will not work for Plant Edition array points.

- `PointSet "PEArrayPointName[10]", val`

Result

`PointSet` sets the element with the index 10 of the Plant Edition array point `PEArrayPointName` to the value of `val`

PointSetMultiple (function)

Syn- tax	<code>PointSetMultiple point1[,point2[,point3...]]</code>
De- scrip- tion	Performs setpoints for up to 30 points in a single setpoint request. If a failure occurs the function returns false, otherwise true is returned.
Com- ments	If you need to set the value of multiple points, use this function rather than issuing multiple single setpoint requests for faster script execution. The point <code>ErrCode</code> property will be set to a non-zero value for a setpoint that failed. The point <code>ErrMsg</code> property will contain the associated error message.

There are two variants of PointSetMultiple. The first variant takes all the points declared in the argument list. The second variant takes an array.

Example 1

This example in Basic demonstrates both variants, argument list and array.

```
Sub Main()
    Dim status As Boolean

    Dim sp1 As New Point: sp1.Id = "SP1"
    Dim sp2 As New Point: sp2.Id = "SP2"
    Dim sp3 As New Point: sp3.Id = "SP3"
    Dim sp4 As New Point: sp4.Id = "SP4"

    sp1.Value = 1
    sp2.Value = 2
    sp3.Value = 3
    sp4.Value = 4

    status = PointSetMultiple(sp1,sp2,sp3,sp4)

    If status = False Then
        If sp1.ErrCode <> 0 Then
            MsgBox sp1.ErrMsg
        End If
    End If

    'r; Using an array

    Dim points(1 To 4) As Point
    Set points(1) = sp1
    Set points(2) = sp2
    Set points(3) = sp3
    Set points(4) = sp4

    status = PointSetMultiple(points)

End Sub
```

Example 2

This example in C# demonstrates only the array variant.

```
using System;
using System.Collections.Generic;
using Proficiency.CIMPLICITY;

public class GetSetNET
{
    public void Main()
    {
        int status;
```

```

Point[] array = new Point[4];

using (Point sp1 = new Point(),sp2 = new Point(),sp3 = new Point(),sp4 = new Point())
    {
        sp1.Id = "SP1";
        sp2.Id = "SP2";
        sp3.Id = "SP3";
        sp4.Id = "SP4";
        array[0] = sp1;
        array[1] = sp2;
        array[2] = sp3;
        array[3] = sp4;
        sp1.Value = 1;
        sp2.Value = 2;
        sp3.Value = 3;
        sp4.Value = 4;
        status = Cimplicity.PointSetMultiple(array);
    }
}

```

PointSetMultipleEx (function)

Syn- tax	<code>PointSetMultipleEx point1[,point2[,point3...]]</code>
De- scrip- tion	Performs setpoints for up to 30 points in a single setpoint request, using the provided setpoint password. If a failure occurs the function returns false, otherwise true is returned.
Com- ments	<p>If you need to set the value of multiple points, use this function rather than issuing multiple single setpoint requests for faster script execution. The point ErrCode property will be set to a non-zero value for a setpoint that failed. The point ErrMsg property will contain the associated error message.</p> <p>There are two variants of PointSetMultiple. The first variant takes all the points declared in the argument list. The second variant takes an array.</p>
Exam- ple 1	This example in Basic demonstrates both variants, argument list and array.

```

Sub Main()

    Dim status As Boolean

    Dim pwd As String

    pwd = "mypassword"

    Dim sp1 As New Point: sp1.Id = "SP1"

    Dim sp2 As New Point: sp2.Id = "SP2"

    Dim sp3 As New Point: sp3.Id = "SP3"

    Dim sp4 As New Point: sp4.Id = "SP4"

    sp1.Value = 1

sp2.Value = 2

sp3.Value = 3

sp4.Value = 4

status = PointSetMultipleEx(pwd,sp1,sp2,sp3,sp4)

    If status = False Then

        If sp1.ErrCode <> 0 Then

            MsgBox sp1.ErrMsg

        End If

    End If

    'r; Using an array

Dim points(1 To 4) As Point

Set points(1) = sp1

Set points(2) = sp2

Set points(3) = sp3

Set points(4) = sp4

status = PointSetMultipleEx(pwd,points)

End Sub

```

Example 2 This example in C# demonstrates only the array variant.

```

using System;

using System.Collections.Generic;

using Proficy.CIMPLICITY;

public class GetSetNET

{

    public void Main()

    {

        int status;

        Point[] array = new Point[4];

```

	<pre> using (Point sp1 = new Point(), sp2 = new Point(), sp3 = new Point(), sp4 = new Point()) { sp1.Id = "SP1"; sp2.Id = "SP2"; sp3.Id = "SP3"; sp4.Id = "SP4"; array[0] = sp1; array[1] = sp2; array[2] = sp3; array[3] = sp4; sp1.Value = 1; sp2.Value = 2; sp3.Value = 3; sp4.Value = 4; status = Cimplicity.PointSetMultipleEx("MyPassword", array); } } </pre>
<p>Error Message</p>	<p>Point.ErrCode Integer value containing the error code for a failed call to PointSetMultiple or PointSetMultipleEx, or zero for a successful operation.</p> <p>Point.ErrMsg String value containing the error message for a failed call to PointSetMultiple or PointSetMultipleEx, or empty string for a successful operation</p>
<p>See Also</p>	<p>PointSetMultiple (function) (on page 914)</p>

SetTimecomponentsHR (function)

<p>Syntax</p>	<pre>SetTimeComponentsHR param1, param2, param 3 ... 9</pre>
<p>Description</p>	<p>Given components of the time. Current time divided into time components of year, month, day, hour, min, sec and nanoseconds.</p>

	Param1	Double. High value of input time.
	Param2	Double. Low value of input time.
	Param3	Integer. Timecomponent.
	Param4	Integer. Timecomponent.
	Param5	Integer. Timecomponent.
	Param6	Integer. Timecomponent.
	Param7	Integer. Timecomponent.
	Param8	Integer. Timecomponent.
	Param9	Long. Nanosecond time component.
Example	<pre> Sub OnMouseUp(x As Long, y As Long, flags As Long) 'Declare variables Dim yy As Integer Dim mm As Integer Dim dd As Integer Dim hh As Integer Dim min As Integer Dim sec As Integer Dim nano As Long Dim qlow As Double Dim qhigh As Double 'Initialize Objects yy = 2011 mm = 7 dd = 13 min = 43 sec = 10 nano = 0 SetTimeFromComponentsHR qhigh,qlow,yy,mm,dd,hh,min,sec,nano MsgBox qhigh MsgBox qlow End Sub </pre>	

See also	GetTimeComponentsHR (on page 864) (function)
----------	--

QINTFromString (function)

Syntax	<code>QINTFromString param1,param2,param3</code>	
Description	To convert one numeric string into QINT , split it's value into 2 doubles and return them.	
	Param1	String.
	Param2	Reference to Double.
	Param3	Reference to Double.
Example	<pre> Sub Main() Dim qlow as Double Dim qhigh as Double Dim qstr as String Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "QINT" ' Set the point id qstr = "1000000899876543212" QINTFromString qstr,qhigh1,qlow1 ret = MyPoint.SetQuadIntValue(qhigh,qlow) End Sub </pre>	
See also	UQINTFromString (on page 923) (function).	

StringFromQINT (function)

Syntax	<code>StringFromQINT param1,param2,param3</code>	
Description	To convert two doubles into one signed 64-bit value and finally to a string	
	Param1	String.
	Param2	Double.

	Param3	Double.
Ex-ample	<pre> Sub Main() Dim qlow as Double Dim qhigh as Double Dim qstr as String Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "QINT" ' Set the point id ret = MyPoint.GetQuadIntValue(qhigh,qlow) qstr = StringFromQINT(qhigh,qlow) 'Get the value as 'string from two doubles qhigh and 'qlow End Sub </pre>	
See also	DoQINTMath (on page 854) (function), DoUQINTMath (on page 855) (function), Point.QuadValueAsString (on page 886) (property, read), Point.QuadValueAsString (on page 887) (property, write), Point.SetQuadIntValue (on page 897) (function), StringFromUQINT (on page 920) (function).	

StringFromUQINT (function)

Syn-tax	StringFromUQINT param1,param2,param3	
De-scrip-tion	To convert two doubles into one signed 64-bit value and finally to a string	
	Param1	String.
	Param2	Double.
	Param3	Double.
Ex-ample	<pre> Sub Main() Dim qlow as Double Dim qhigh as Double Dim qstr as String Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "QINT" ' Set the point id </pre>	

	<pre> ret = MyPoint.GetUQuadIntValue(qhigh,qlow) qstr = StringFromUQINT(qhigh,qlow) 'Get the value as 'string from two doubles qhigh and 'qlow End Sub </pre>
See also	DoQINTMath (on page 854) (function), DoUQINTMath (on page 855) (function), Point.QuadValueAsString (on page 886) (property, read), Point.QuadValueAsString (on page 887) (property, write), Point.SetQuadIntValue (on page 897) (function), StringFromQINT (on page 920) (function).

Trace (statement)

Syntax	Trace a\$
Description	Traces (prints) a string to the trace output. By default, when running in the Program Editor, tracing will be output to the trace window. When running from the Event Manager, tracing must be specifically enabled (TraceEnable) in order for tracing to occur.
Example	<pre> Sub Main() Dim x as new Point a\$ = InputBox\$("Enter a point id") x.Id = a\$ x.OnChange top : x.GetNext Trace str\$(x.TimeStamp) & " " & x.Value goto top End Sub </pre>

TraceEnable/TraceDisable (statement)

Syntax	TraceEnable file\$ TraceDisable
--------	---

De- scrip- tion	<p>TraceEnable enables tracing to a file. The file will be located in your project's log directory. Tracing to a file is only supported from the event manager. The trace output will be written to the log directory. Tracing has a performance impact since the file is opened and closed for each write. Tracing is intended for debug use only and should be removed from production code.</p>
	<p>TraceDisable disables tracing to a file</p>
Ex- am- ple	<pre> Sub Main() if PointSet("TRACE_TRIGGER") = TRUE then TraceEnable "MY_LOG" end if Trace "Trace Message 1" Trace "Trace Message 2" TraceDisable End Sub </pre>

UQINTFromString (function)

Syntax	UQINTFromString param1,param2,param3	
De- scrip- tion	To convert one numeric string into UQINT , take a positive value with the highest value that can be taken by ULONGLONG and return it.	
	Param1	String.
	Param2	Reference to Double.
	Param3	Reference to Double.
Exam- ple	<pre> Sub Main() Dim qlow as Double Dim qhigh as Double Dim qstr as String Dim MyPoint as new Point ' Declare the point object MyPoint.Id = "QINT" ' Set the point id qstr = "1000000899876543212" UQINTFromString qstr,qhigh1,qlow1 ret = MyPoint.SetQuadIntValue(qhigh,qlow) End Sub </pre>	

See also	QINTFromString (<i>on page 920</i>) (function)
----------	--

Chapter 5. Basic Control Engine User Interface

About the BCEUI

Use the Basic Control Engine User Interface (BCEUI) to connect to CIMPLICITY projects in your enterprise and monitor events. With this user interface, you can:

- View the status of actions executed by selected events in various projects.
- Pause, resume, and stop scripts executed by events.
- Manually trigger events.
- Configure a view of projects and events and save the configuration in a file for recall.

The BCEUI window displays the status of actions triggered by events that are currently being monitored by BCEUI. You can use the Paused option to display this list in dynamic or paused mode.

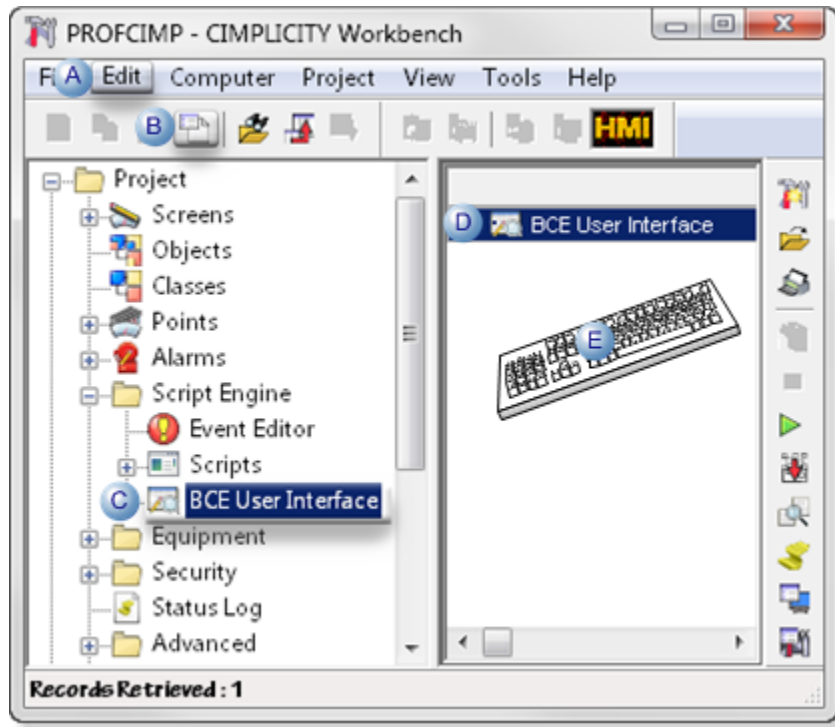
- In dynamic mode, the list is automatically refreshed as events occur or change status.
- In paused mode, the list remains fixed until you update it. To update the list, you can select Refresh from the View menu, or press F5.

Note the following about the display:

- Actions for all running projects that BCEUI is connected to are displayed in black.
- If BCEUI is connected to a CIMPLICITY project and monitoring events, and the project stops:
 - All events for the project are grayed out in the Properties dialog box.
 - Triggering is disabled for events in the stopped project.
 - A \$Disconnected event displays in the main window with a message telling you which project is stopped. This event runs and tries to reconnect to the project until either the project starts or you close your BCEUI session.
- All unfinished actions in the main window are grayed out to indicate that their current status is unknown.
- When a CIMPLICITY project that BCEUI is attempting to connect to restarts, grayed actions are redisplayed in black and refreshed to their current status.

Open the BCEUI Window

1. Select **Project>Basic Control Engine>BCE User Interface** in the Workbench left pane.
2. Select **BCE User Interface** in the Workbench right pane.
3. Do one of the following.



A Click Edit>Properties on the Workbench menu bar.	
B Click the Properties button on the Workbench toolbar.	
C In the Workbench left pane: a. Right-click BCE User Interface . b. Select Properties on the Popup menu.	
D In the Workbench right pane:	
Either	Or
Double click BCE User Interface .	a. Right-click BCE User Interface . b. Select Properties on the Popup menu.
E Press Alt+Enter on the keyboard.	

4. Right-click **BCE User Interface**.
5. Select Properties on the Popup menu.
6. Right-click **BCE User Interface**.
7. Select Properties on the Popup menu.

BCEUI Menus

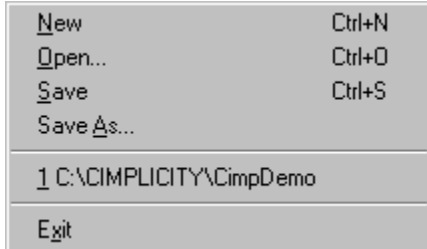
BCEUI Menus

You can use the menu options to save and restore event monitoring configurations, add or list events, pause, stop or resume scripts, trigger events, pause and resume dynamic updates, refresh the display and access Help.

The menus are:

File menu
Events menu
Scripts menu
View menu
Help menu

BCEUI File Menu



The File menu functions are:

New	Creates a new BCEUI document.
Open	Opens an existing BCEUI document in your currently active BCEUI window.
Save	Saves the current BCEUI document to a file.
Save As...	Saves the current BCEUI document to a file. Use this option if you want to specify the path-name of the saved file.
Recent File	Displays a list of recently opened BCEUI document files for easy retrieval.
Exit	Exits the CIMPLICITY BCEUI viewer.

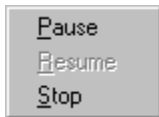
BCEUI Events Menu



The Events menu functions are:

List	Opens the Properties dialog box, from which you can add, delete or trigger events.
Add	Opens the Select an Event browser, from which you can connect to a project and select events to add to the list of monitored events.

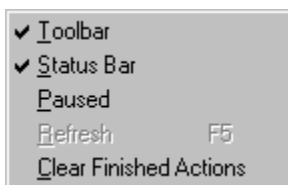
BCEUI Scripts Menu



The Scripts menu functions are:

Pause	Pauses any currently selected running scripts.
Resume	Resumes any currently selected paused scripts.
Stop	Stops any currently selected scripts that are paused or running.

BCEUI View Menu

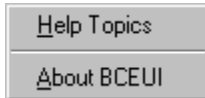


The View menu functions are:

Toolbar	Enables/disables display of the Toolbar.
Status Bar	Enables/disables display of the Status Bar.

Paused	Toggles between dynamic and paused view.
Refresh	Updates the paused view.
Clear Finished Actions	Clears finished actions from the event list.

BCEUI Help Menu



The Help menu functions are:

Help Topics	Displays the main Help windows for the BCEUI.
About BCEUI	Displays the program identification, version number and copyright for the BCEUI.

BCEUI Window Pop-up Menu

1. Select a running or paused script.
2. Press the right mouse button.

BCEUI Toolbar

You can use the Toolbar option on the View menu to turn on and off the display of the BCEUI Toolbar. You can fix the Toolbar in the BCEUI window or display it in a separate window at your discretion.

The buttons on the BCEUI Toolbar are:

	New	Creates a new BCEUI document.
	Open	Opens an existing BCEUI document.
	Save	Saves the current BCEUI document to a file.
	Event List	Opens the Properties dialog box, from which you can add, delete or trigger events.
	Add Events	Opens the Select an Event browser, from which you can connect to a project and select events to add to the list of monitored events.

	Stop Scripts	Stops any currently selected scripts that are paused or running.
	Pause Scripts	Pauses any currently selected running scripts.
	Resume Scripts	Resumes any currently selected paused scripts.
	Pause View	Toggles between dynamic and paused view.
	Clear Finished Actions	Clears finished actions from the view.
	About	Displays the program identification, version number and copyright for the BCEUI.

BCEUI Shortcut Keys

The following are the more commonly used keystrokes that are available for your use in the BCEUI:

Ctrl+N	Creates a new BCEUI view.
Ctrl+O	Opens an existing BCEUI document.
Ctrl+S	Saves the current BCEUI document to a file.
F5	Updates the paused view.
F1	Opens the Help window for the BCEUI.

BCEUI Viewer

BCEUI Viewer

To create a BCEUI view, you need to:

- Use the Select an Event browser to connect to a project and select events to add to the BCEUI event list.
- Use the Properties dialog box to list monitored events, add or remove events from the view, and trigger events manually.

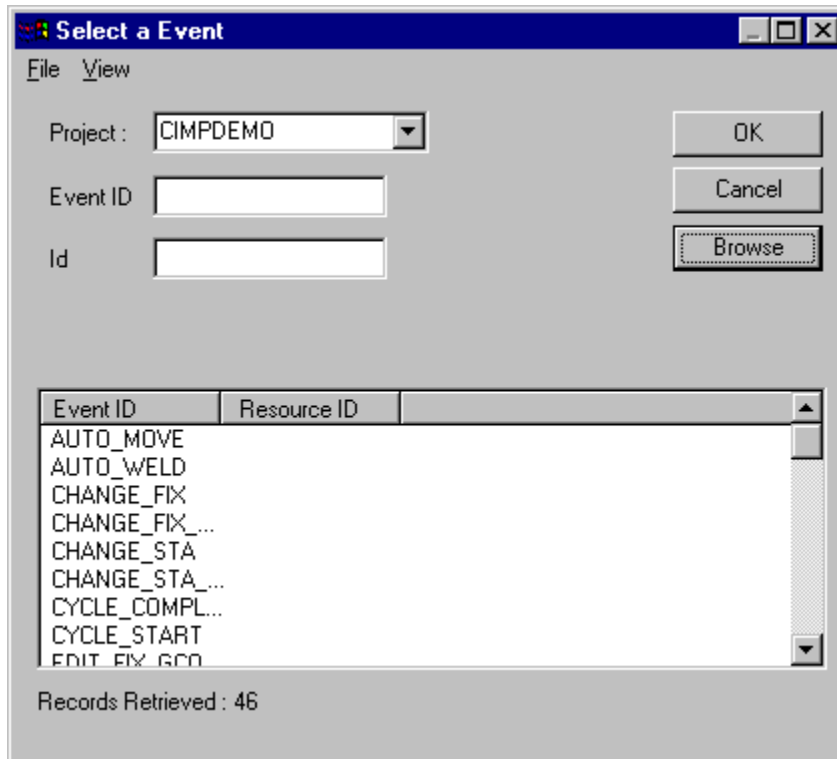
After you create a BCEUI view, you can select script actions and pause, resume, or stop the scripts.

Once you create a BCEUI view, you can save it. You can recall saved views at any time.

1 (on page 931)	Select events.
2 (on page 932)	Toggle the auto browse.
3 (on page 933)	Connect to a project.
4 (on page 933)	Select events.
5 (on page 933)	Use the event list.
6 (on page 935)	Set the maximum number of completed actions.
7 (on page 935)	Add events to the View.
8 (on page 935)	Remove events from the view.
9 (on page 935)	Trigger events.

1. Select Events in the Browser

When you select Add from the Events menu or click the Add Events button on the Toolbar, the Select an Event browser opens.



From the Select an Event browser, you can:

- Enable/disable Auto Browse.
- Change the display attributes.
- Connect to a project.
- Select events from the project for monitoring.

After you select events and click **OK**, the Properties dialog box automatically opens so that you can add the selected events to your view. If you click **Cancel**, the Select an Event browser closes and the main BCEUI window is redisplayed.

2. Toggle the Auto Browse

By default, the Auto Browse option is disabled. If you enable the Auto Browse option, whenever you open the Select a Event dialog box, the events for the first project in the **Project** list are automatically displayed in the list window.

If Auto Browse is enabled, a check mark is displayed to its left in the View menu.

1. Select the View menu.
2. Select the Auto Browse option.

3. Connect to a Project

1. Click the drop-down list button to the right of the **Project** field to see the list of currently available projects.
2. Select a project from the list.
3. Click **Browse** to see the list of events available for the project.

4. Select Events

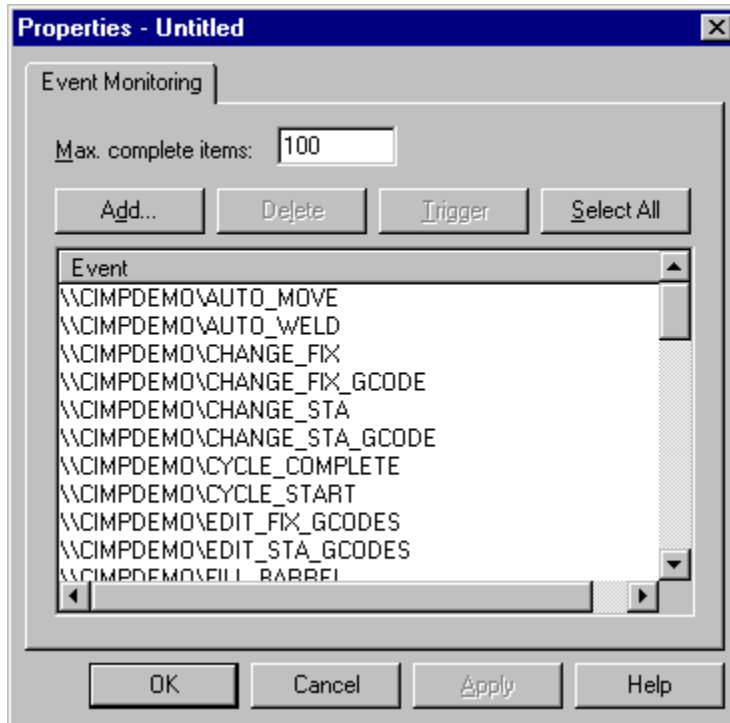
1. Highlight the events you want to select. You may use the **Shift** and **Ctrl** keys when selecting multiple events.
2. Click **OK** to transfer your selection to the BCEUI event list and close the Select an Event browser.

5. Use the Event List

Do one of the following to open the Properties dialog box.

- Select List from the Events menu, or
- Click the **Event List** button on the toolbar, or
- Click **OK** on the Select a Event browser after selecting events.

Result: The Properties dialog box opens.



Use this dialog box to:

- Set the maximum number of completed actions to be displayed by the view.
- Add events to the monitored list.
- Delete events from the monitored list.
- Trigger events.



Note:

Note the following:

- Triggering is enabled only for events in connected projects that are running.
- All events for projects that are running and BCEUI is connected to are displayed in black.
- Events in the list that belong to projects that are not currently running or that become disconnected are grayed out.
- When you add events for a new project, they are grayed out in the Properties dialog box because BCEUI has not connected to the project yet.

- The first time you select an event for a newly selected project, then select **Apply**, BCEUI connects to the project. When the connection completes successfully, all the events for the project are displayed in black.
- You can select events for projects that are not currently running or that are disconnected. When the project starts, BCEUI will automatically connect with the project and start monitoring the events.

6. Set the Maximum Number of Completed Actions

The default maximum number of completed actions that the BCEUI window can display is 100. You can choose less or more than this number. Once the list reaches its maximum, the oldest completed action is removed when the newest one is added.

1. Enter the number in the **Max. complete items** field.
2. Click **OK** or **Apply**.

7. Add Events to the View

1. Select the events you want to monitor from the list of events in the Select an Event browser. You can use the **Shift** and **Ctrl** keys to select multiple events.
2. Click **Apply** to add the events to the view and keep the Properties dialog box open, or click **OK** to add the events to the view and close the Properties dialog box.

8. Remove Events from the View

1. Select the events in the list that you want to remove. You can use the **Shift** and **Ctrl** keys to select multiple events. You can also click **Select All** to select all events in the list.
2. Click **Delete**.

The events you select are removed from the BCEUI window and the Properties dialog box.

They will not appear again in Properties dialog box until you add them in the Select a Event browser, and they will not be monitored again in the BCEUI window until you select them for viewing in the Properties dialog box.

9. Trigger Events

Your role must have the Trigger Event privilege enabled for you to be able to trigger events for a particular project.

1. Select the events in the list that you want to trigger. You can use the Shift and Ctrl keys to select multiple events. You can also click Select All to select all events in the list.
2. Click Trigger.

The Confirm Trigger Action message box opens and displays the first event to trigger.

3. Click one of the following

Yes to All	Trigger all the selected events.
Yes	Trigger this event.
No	Cancel the trigger for this event.
Cancel	Cancel your request



Note:

If you click Yes or No and you are triggering multiple events, you are automatically prompted to confirm the next trigger action.

The statuses of the events you trigger are displayed in the BCEUI window.

Control Scripts

Control Scripts

Your role must have the Script Control privilege enabled for you to be able to pause, resume and stop scripts in the BCEUI window in specific projects.

You can do the following:

- Pause running scripts. (Only Basic scripts)
- Resume paused scripts. (Only Basic scrips)
- Stop running or paused scripts. (All scripts)

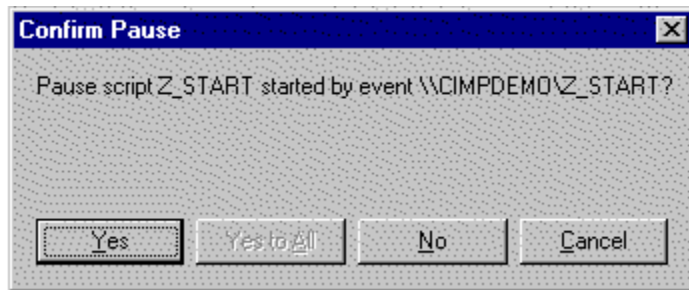
Pause Scripts

1. Select the actions whose scripts you want to pause in the BCEUI window. You can use the **Shift** and **Ctrl** keys to select multiple actions.

You may safely select multiple scripts, even if some of the scripts you select cannot be paused (such as stopped scripts or scripts that are already paused). Such scripts will not be affected by the **Pause Scripts** request.

2. Do one of the following.
 - Select Pause from the Scripts menu, or
 - Click the **Pause Scripts** button on the toolbar, or
 - Click **Pause** from the Window Pop-up menu.

The Confirm Pause dialog box opens.



3. Click one of the following.

Yes to All	Pause all the selected scripts.
Yes	Pause this script.
No	Cancel the pause request for this script.
Cancel	Cancel your request.



Note:

If you click **Yes** or **No** and you are pausing multiple scripts, you are automatically prompted to confirm the next script in the list.

Resume Scripts

1. Select the actions whose scripts you want to resume in the BCEUI window. You can use the **Shift** and **Ctrl** keys to select multiple actions.
2. Do one of the following.
 - Select Resume from the Scripts menu, or
 - Click the **Resume Scripts** button on the toolbar, or
 - Select Resume from the Window Pop-up menu.

The Confirm Resume dialog box opens.

3. You may select one of the following.

Yes to All	Resume all the selected scripts.
Yes	Resume this script.
No	Cancel the resume request for this script.
Cancel	Cancel your request.

Note: If you click Yes or No and you are resuming multiple scripts, you are automatically prompted to confirm the next script in the list.

Stop Scripts

1. Select the actions whose scripts you want to stop in the BCEUI window. You can use the Shift and Ctrl keys to select multiple actions.
2. Do one of the following.
 - Select Stop from the Scripts menu, or
 - Click the Stop Scripts button on the toolbar, or
 - Select Stop from the Window Pop-up Menu.

The Confirm Stop dialog box opens.

3. Click one of the following.

Yes to All	Stop all the selected scripts.
Yes	Stop this script.
No	Cancel the stop request for this script.

Cancel	Cancel your request.
--------	----------------------

Note: If you click **Yes** or **No** and you are stopping multiple scripts, you are automatically prompted to confirm the next script in the list.

The status of the scripts for the events you select changes from Paused or Running to Stopped and the message field for each stopped script displays the line number where the script was stopped.



Note:

Once you stop a script, you cannot restart it with the **Resume** command.

Chapter 6. Action Calendar

About the Action Calendar

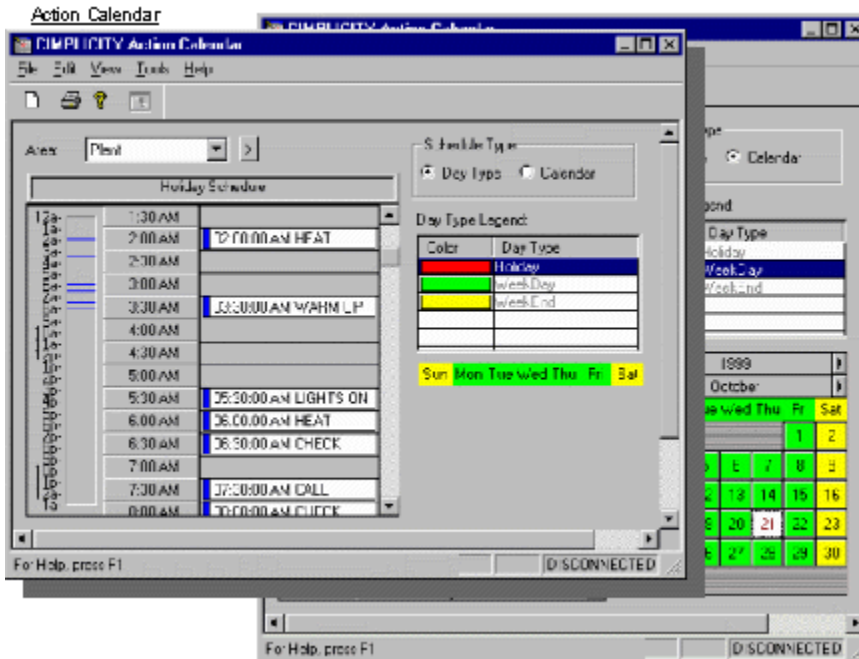
Action Calendar is a feature added to CIMPLICITY, which allows you to dynamically create, maintain, and execute a calendar schedule of manufacturing events and associated actions. Turn on lights, heat, and equipment based on a schedule, which you configure and maintain through simple point and click actions.

This Application Module is fully integrated with CIMPLICITY software's Base System functionality to enhance its already powerful monitoring capability in a full range of computer integrated manufacturing environments. Designed from the ground up as a true client / server architecture, CIMPLICITY has always provided more than simple monitoring and control. CIMPLICITY software's flexible system architecture and modular design also allows for easy add-on of functionality. When you take on the challenge of an enterprise wide system, you face challenges which simple MMI systems just cannot handle. With the CIMPLICITY Action Calendar you can coordinate plant operations on a timed basis.

The Action Calendar Application Module, which interfaces with the Base System Point Management facility and User Interface, allows you to easily schedule the execution events in your system through a simple calendar based user interface. Configured events can drive real world I/O through CIMPLICITY and turn equipment/utilities on and off based on production schedules. In addition internal events can be activated to trigger:

- Data collection
- Data logging
- Report generation
- Execution of scripts or programs.

Managing events and activities associated with your production schedule are made easy with the CIMPLICITY Action Calendar.



Planning for the Action Calendar

What the Action Calendar Does

The Action Calendar option gives you, the system administrator, the ability to build a set of automated events that can be applied in one area of or throughout your plant. You can invoke events with associated actions at specific times of a day and during day types (for example, weekdays) that you define.

An action can be any action supported by the Event Editor. For example an action can set a point, generate an alarm, download a recipe or even run a user written Basic Script.



Note:

The Action Calendar schedules events with associated actions. It is not designed for Production scheduling.

The Action Calendar has two major components. The:

1. [Graphical User Interface \(on page 942\)](#) (GUI) allows users to interactively configure and view schedule information.
2. [Scheduler \(on page 945\)](#) is responsible for ensuring that your operations are initiated at the appropriate times.

Action Calendar's Graphical User Interface

The Graphical User Interface, which has a familiar electronic day planner appearance:

- Provides the screens needed for you to configure all Calendar data, including areas, event actions, and schedules.
- Lets you project what events are scheduled either today or for any date in the future, based on existing Action Calendar configuration data.
- Lets you specify exceptions to these standard schedules, as needed, to meet production needs for a specific date. These schedule overrides can be used to completely alter a day's schedule (for example, to accommodate holidays), or to modify, add, or skip a single event on a specified date.

Action Calendar's Scheduler

The Action Calendar's Scheduler is a CIMPLICITY resident process that:

- Determines the daily production schedules by:
 3. Combining all standard events for the current date.
 4. Applying all overrides associated with that date, until the total plant wide schedule has been calculated.
- Initiates events based on these schedules.
- Performs periodic cleanup of the Action Calendar configuration data so that outdated information (in particular, overrides that correspond to dates in the past) is automatically purged from the system.

When to use Other CIMPLICITY Tools

1. If you want to schedule an event to execute every minute, use the CIMPLICITY Event Editor. The Event Editor provides an easier way than the Action Calendar to schedule these types of repetitive events.
2. If you want to schedule an event in less than one-second real time intervals, use a PLC or a CIMPLICITY PC control to perform the real time control. In the Action Calendar, events can be scheduled to the second. On a normally loaded system, your event will execute within +1 second of the target time.

Action Calendar Interface Overview

It is through the Action Calendar interface that you:

[Create and define Areas with scheduled events \(on page 943\)](#)

- [Project Schedules for each Area \(on page 944\)](#)

Action Calendar Areas in a Facility

The Action Calendar lets you divide your facility into any number of areas, each with its own unique set of events and schedules.

An Area defines specific locations, stations, or work units with which schedule information may be associated. Examples of areas are

- ASSEMBLY
- Inspection
- Factory

Each area has its own configuration data. One immediate benefit of this feature is that users only need to be familiar with their own area, and do not need to understand the entire plant's operation in order to create or modify schedules. Of course, you, the system administrator, may also define plant wide areas, to schedule events for the entire factory.

For example, you may dedicate an area to `factory_lights`. You can then create a configuration that will instruct the Action Calendar to turn the lights on and off throughout the plant, at the times you designate.

The information required for an area includes definitions for:

- Events
- Day types with associated weekdays
- Schedules for the day types

Event Definitions

An [Event definition \(on page 947\)](#) provides a list of actions to perform such as Setpoints, and assigns a unique event name to the association.

Example

To define the event `MAIN_LIGHTS_ON`, associate the CIMPLICITY Point ID `MASTER_LIGHTS` with a value of one (1).

Day Type Definitions

Day type definitions (on page 969) (or classifications) identify the different types of days are required to accommodate the various production needs of the plant.

Each day type within an area

- Has its own unique schedule of events
- Includes the days of the week (for example, Tuesday) that you designate
- Cannot have a day of the week (for example, Wednesday) that is assigned to a different day type

Examples of day types with assigned days include

Area A

- **Weekday:** Monday, Tuesday, Wednesday, Thursday, Friday
- **Weekend:** Saturday, Sunday

Area B

- **Workday.** Tuesday
- **Maintenance.** Friday

Schedules

A [Schedule definition \(on page 982\)](#) specifies the sequence and timing of events associated with the area. All times may be specified to the nearest second on a 24-hour clock.

Example

The schedule for a FACTORY area specifies the event MAIN_LIGHTS_ON to occur at 6:00:00am.

Projected Schedules for the Action Calendar

Projected Schedules

[Projected schedules \(on page 982\)](#) display a time ordered list of events that are scheduled to occur on a selected date.

In addition to building base schedules for each area, the Graphical User Interface provides you with the mechanism to:

- Project what an area's schedule will look like for today or for any date in the future.
- Override any or all events associated with that date with:
- Day type overrides completely replace the set of events associated with one-day type with the base schedule associated with a new day type.
- Event overrides affect individual events, letting you dynamically add, skip, or reschedule events at any time.

About the Action Calendar Scheduler

1. Determines the current day of the week.
2. Determines, for each area, the appropriate configured day type.

This day type will be either the standard day type for the area or, if the day type has been overridden, the new requested day type.

3. Reads that day type's schedule, along with any event overrides specific to today's schedule, and merges with all other area schedules for that day.
4. Begins processing of the events once the complete schedule has been built.



Important:

Some important notes about how the Action Calendar Scheduler deals with events include:

- If any events are being scheduled during the time that the Action Calendar is actually generating the schedule, these events will be initiated, in order, immediately upon completion of the schedule generation.
- There will be no predictable order for events that are scheduled to execute simultaneously.
- If a single event performs more than one action, the sequence of the actions is guaranteed.

When the Calendar determines that an event needs to be initiated, it sends an event request to the Event Manager (EM) subsystem. If the event has been configured with logging enabled, then the success or failure of the actions will be logged.

Action Calendar Planning Configuration

Action Calendar Planning Configuration

1. Areas
2. Events with associated actions
3. Day types with assigned days

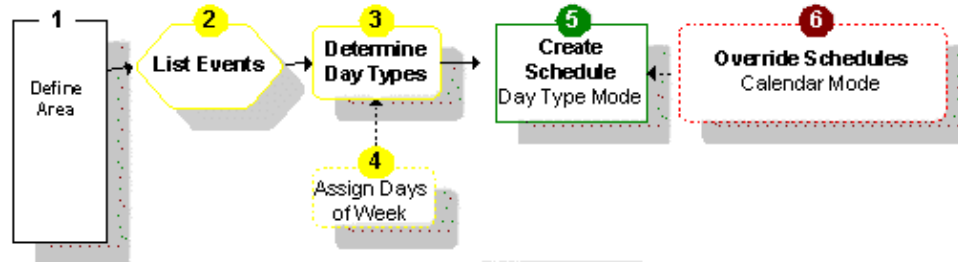
4. Actual Schedules

Once you have completed the overall plan, you can then:

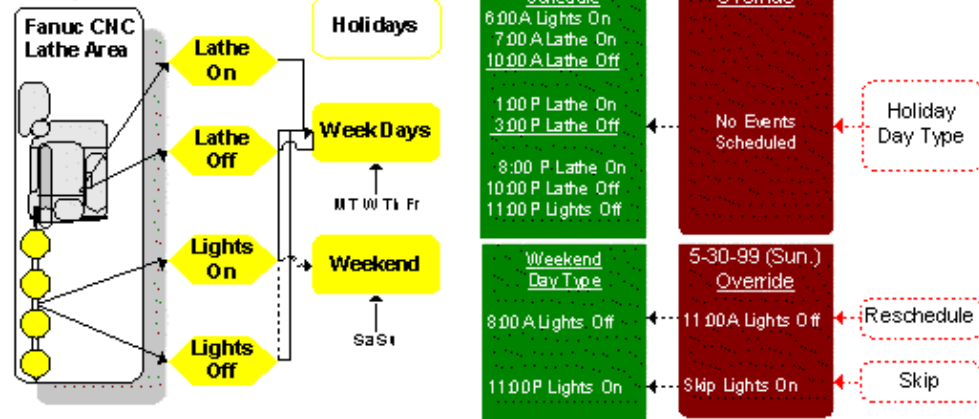
- 5. Make one-time adjustments to any of the schedules
- 6. Expedite schedule adjustment through Offset Events

Simple Planning for an Area Called Fanuc CNC Lathe

Procedure



Example



Setting up Areas

Setting up Areas

When setting up your configuration you need to decide what are the:

- Areas into which you will group events.
- Events you will schedule in each area.

Definition of Areas to Group Action Calendar Events

You may want to begin planning by defining what appear to be obvious areas. As you continue planning your configuration, you may see different relationships that prompt you to redefine your original areas.

The key to defining areas is to identify one or more:

- **Physically independent** areas that have their own repetitive actions in machinery or peripheral equipment, for example, a raw material cutting area that has its own light source.
- **Logical** areas, each with its scheduling needs, for example, reporting that is due every Friday.



Note:

The Action Calendar lets you handle your entire facility through a single area. In fact, Action Calendar provides a pre-configured area, **Plant that** you can use as a starting point.

A plant wide area may be appropriate for company wide actions. However, most likely you will also need to define more specific areas.

Definition of Events Scheduled for Action Calendar Areas

Once you have your initially defined areas, review each area to determine the types of operations that occur in that area. There may be one or more events associated with each operation and one or more actions associated with each event.

Each operation typically corresponds to something that is controllable through a CIMPPLICITY point.

In most cases, there will be a set of two or more events (or ultimately, event point values) that correspond with the operation point referenced in your configuration.

Events can be:

- Actions which enable and disable a digital point
- A series of actions, each of which sets analog or text points to one of multiple values
- Scripts
- Alarms
- Recipes



Note:

The Action Calendar lets you configure events for points which have been defined but which have not been incorporated into the runtime system via a Configuration Update operation.

This enables you to set up a system for a future point configuration modification that will make these events valid at a future date. Until that time, these events will be ignored by the Calendar, and messages will be logged whenever an attempt is made to incorporate these events into a day's schedule.

Example

A machine requires a light to be on for part of the day and to be off for the remainder.

1. Create an operation point for the light, called **Machine_Light**.
2. Configure two events to control **Machine_Light**. Call the events

unit_on

unit_off .

The Action Calendar will turn the machine light on and off, based on the times for which you schedule **unit_on** and **unit_off**.

You may also have a series of actions that involve turning off the machine light, which you can call up through the use of a script.

Setting up Day Types with Assigned Days

Setting up Day Types with Assigned Days

Having determined at least the initial set of events that will be required within an area, you can begin to determine when these events may be invoked. As you review the day types, you may need to redefine some areas.

The Action Calendar carries out events that you schedule for day types.

You may:

- Require multiple standard day types to handle the scheduling requirements such as weekdays, weekends and holidays.
- Configure additional reserved day types for a specific area because your plant has other conditions, which you must accommodate (for example, extended production days or shutdown).
- Configure a day type that has no events. Any days of the week assigned to that day type will have no events scheduled for that area.

However, within one area, you can only assign a specific day of the week, for example, Friday, to one day type.

Example of Assigning Days of the Week to Day Types

Example

In one area you:

1. Define Weekdays as a day type to which you assign Monday through Friday
2. Decide that the area has additional events on Friday and, therefore, define Friday as a day type. You assign Friday to it, thereby removing Friday from Weekdays
3. Do define Saturday and Sunday as None. "None" has no schedule.

The Action Calendar will display

- Weekdays as Monday through Thursday
- Friday as Friday
- Saturday and Sunday are None.

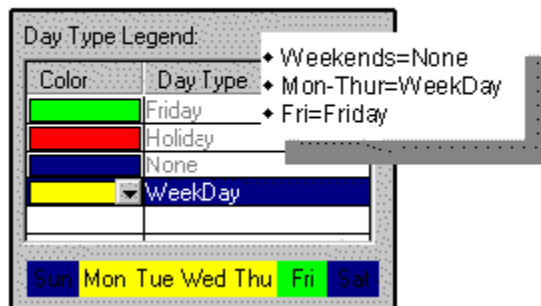
At this stage of your planning, you may decide to:

- Create another area where you:
- Create Friday as a day type and schedule the additional events,
- Keep the day type Weekdays as Monday through Friday in this area.

or

- Keep Friday as a day type in this area with a schedule that includes all the events for Weekdays and Friday's additional events. Action See "Configuring the Action Calendar – Copying Day Types"

Days Assigned to Day Types Example



Making One Time Adjustments to Schedules

Making One Time Adjustments to Schedules

Once you have designed and configured standard configuration data, you can analyze your system's needs for any date specific overrides. Overrides supported by the Action Calendar can be divided into two distinct categories:

They are:

- Day type overrides.
- Event overrides.



Note:

Since the configuration screens are intended to accommodate standard, repetitive events, exception conditions should not be included in the initial schedules. Instead, these can be more appropriately handled through event or day type overrides, discussed in more detail below.

Day Type Overrides Definition

Day type overrides replace an entire day's schedule for the designated area.

This is particularly useful in the case of holidays or special events that fall on a day in which production would normally run.

Example

If July 4 falls on a Monday, which is configured as a Weekday in the base configuration, you use a day type override to define July 4 as a Holiday.

Another example is a case where, due to increased production demands, the plant decides to run extra hours to meet these demands. If an alternate schedule and day type have been configured to meet these additional production hours, you can use the alternate day type to override the standard day type.

Event Overrides Definition

Event overrides are specific to a single event.

The three supported event overrides are:

1. Adding an unscheduled event to the schedule.

This type of override is useful in situations when an operation is required on a given date where it normally would not be part of that day's schedule.

2. Rescheduling a specific event to a new time.

This override can be used to alter the time at which a scheduled event is initiated.

3. Skipping an event, which would otherwise be initiated.

This override is useful in situations where a standard event is not desired on a particular date.

There are a number of rules associated with overrides, which are important to understand.

- Any override, which is scheduled for the current day, will be incorporated into the current schedule immediately upon being requested by the user.
- If a day type override is configured, all existing event overrides for that same date and area will be immediately deleted, so that an override is not inadvertently applied to an event for which it was not intended.
- Initiate all day type and event overrides after you project a schedule for the date in question. These overrides will be automatically reflected in the projected schedule being presented to the user.

Expediting Schedule Adjustment through Offset Events

As you define an area's events and their scheduling requirements, including needs for overrides, you may discover groups of events that are logically related and are, therefore, always processed as a set.

You can expedite scheduling the group at any time, by establishing an Initial/Offset Event relationship among these events.

An Initial Event is one that, whenever scheduled, has one or more offset events scheduled static to it at fixed time intervals.

An offset event occurs static to when the initial event occurs. For example, it may occur one minute before the initial event, or three hours after.

Establishing an initial/offset event relationship has the following impact on a schedule, no matter whether you are adding it to the base schedule or as an event type override.

- Adding an unscheduled initial event to a schedule causes all the initial event's offset events to be scheduled automatically.
- Rescheduling an initial event to a new time causes all the initial event's offset events to be rescheduled automatically, so that the fixed time intervals between all the events in the set are preserved.

- Skipping or deleting an initial event, which would otherwise be initiated, causes all the initial event's offset events to be skipped or deleted automatically.
- Changing the interval of the offset events causes all instances of the scheduled event/offsets to change automatically.

You uniquely define Initial/offset event relationships for each area. Within an area, the relationships you define apply across schedules for all day types.

Production Shifts and Days

Production Shifts and Days

If your plant has multiple shifts and 24 hour or nonstandard production days, you may have to customize the Action Calendar definitions of production shifts and days.

Changing Production Shift Parameters

If your production facility operates in a multiple shift environment, it may be desirable from an operational or maintenance viewpoint to configure your system so that each shift maintains its own schedule of operations. Since an Action Calendar area may only be associated with a single day type at any given time, you can define Pseudoareas within an Action Calendar.

A pseudoarea maintains its own event, day type, schedule, and day of the week definitions, while still merging the individual schedules at runtime into a single plant wide schedule for any given day.

Within a given area, each Pseudoarea will really represent the same set of devices, locations, points, etc., but will only schedule events against these points during the respective timeframe of each shift.

Example

Pseudoareas could be "Assembly_Shift1" and "Assembly_Shift2", each with their own respective schedules.

Modifying Production Days

By default, Action Calendar is set up to display a production day as midnight this morning through midnight tonight.

However, if your production facility has production days that run 24 hours, with one production shift running through midnight, you may want to set up different parameters for your production day.

If you prefer, you can configure the Action Calendar to adjust the 24hour-production period to a period more desirable for your production facility.

Configuration Changes Incorporated into the System

As you configure schedules through the Action Calendar User Interface, the data is sent to the Action Calendar, which stores it in a set of Action Calendar configuration files. The data is immediately incorporated into the runtime system. This means that if you modify a day type schedule that is in effect for the day, your modifications will be applied immediately to the currently running schedule.

When you have to add an event to the current day's schedule you may add it to the schedule using an [Add Event \(on page 950\)](#) override.

Sample Factory Configuration Example

Sample Factory Configuration Example

1. Runs the same schedule five days a week.
2. Does not run manufacturing on the weekends.
3. Has maintenance crews that work seven days a week.

The Action Calendar is configured for:

- Production day schedule
- Points
- Areas
- Events
- Assembly events
- Offset events
- Day types
- Schedules
- Day of the week assignments

Sample Factory Production Day Schedule

The schedule for a production day looks like:

Activity	Time	Area of Plant Affected
Factory Lights On	5:30 AM	ALL
Kilns Turned On	5:30 AM	Kilns Only
Kilns Warmed Up	5:45 AM	Kilns Only
Kilns On Full	6:00 AM	Kilns Only
Assembly Conveyors On	6:00 AM	Assembly Only
Morning Break Starts	10:30 AM	ALL
Morning Break Ends	10:45 AM	ALL
Lunch Starts	12:00 PM	ALL
Lunch Ends	12:45 PM	ALL
Afternoon Break Starts	2:15 PM	ALL
Afternoon Break Ends	2:30 PM	ALL
First Shift Ends	3:45 PM	ALL
Assembly Conveyors Off	3:45 PM	Assembly Only
Assembly Conveyors On	4:00 PM	Assembly Only
Second Shift Starts	4:00 PM	ALL
Dinner Starts	5:30 PM	ALL
Dinner Ends	6:15 PM	ALL
Evening Break Starts	8:00 PM	ALL
Evening Break Ends	8:15 PM	ALL
Assembly Conveyors Off	11:00 PM	Assembly Only
Kilns Cooled Off	11:00 PM	Kilns Only
Kilns Turned Off	11:15 PM	Kilns Only
Factory Lights Off	11:30 PM	ALL

Sample Factory Point Configuration

In order to control production activities, a set of CIMPLICITY Point IDs is needed as follows:

Point ID	Point Type	Function
MAIN_LIGHTS	DIGITAL	Used to control factory lights
KILN_ENABLED	DIGITAL	Used to enable/disable kilns
KILN_TEMP	ANALOG	Used to control kiln temperature
ASSY_CONVEY	DIGITAL	Used to enable/disable conveyors
BREAK_LIGHT	DIGITAL	Used to turn on/off light to signal breaks



Note:

: This example assumes that these points are already configured.

Sample Factory Area Configuration

1. PLANTWIDE
2. KILN_AREA
3. ASSEMBLY

Sample Factory Event Configuration

Having defined three areas, we can now split the list of events (as indicated above) into area specific events, and begin to configure each area.

PLANTWIDE Events

Beginning with the PLANTWIDE area, we can define the following events:

Event ID	Point ID	Point Value
LIGHTS_ON_EV	MAIN_- LIGHTS	1
LIGHTS_OFF_EV	MAIN_- LIGHTS	0
START_BREAK_EV	BREAK_LIGHT	1
END_BREAK_EV	BREAK_LIGHT	0
START_LUNCH_- EV	BREAK_LIGHT	1
END_LUNCH_EV	BREAK_LIGHT	0

KILN_ONLY Events

The events associated with the KILN_ONLY area are:

Event ID	Point ID	Point Value
KILN_ON_EV	KILN_ENABLED	1
KILN_OFF_EV	KILN_ENABLED	0

KILN_WARM_EV	KILN_TEMP	250
KILN_FULL_EV	KILN_TEMP	500
KILN_COOL_EV	KILN_TEMP	100

Sample Factory Assembly Event Configuration

Finally, the events associated with the ASSEMBLY area are:

Event ID	Point ID	Point Value
CONVEYOR_ON_EV	ASSY_CONVEY	1
CONVEYOR_OFF_EV	ASSY_CONVEY	0

Sample Factory Offset Event Configuration

Having defined events for each area, we can now create sets of events that are done in a group. Creating events offset from a base event by a static time does this.

PLANTWIDE Offset Events

Beginning with the PLANTWIDE area, we can define the following offset events:

Base Event	Offset Event	Offset Time
START_BREAK_EV	END_BREAK_EV	00:15
START_LUNCH_EV	END_LUNCH_EV	00:45

KILN_ONLY Offset Events

The offset events associated with the KILN_ONLY area are:

Base Event	Offset Event	Offset Time
KILN_ON_EV	KILN_WARM_EV	00:15

KILN_ON_EV	KILN_FULL_EV	00:30
KILN_OFF_EV	KILN_COOL_EV	00:15

Sample Factory Day Type Configuration

There is one set of day types for the PLANTWIDE Area and one for the KILN_ONLY and ASSEMBLY areas.

PLANTWIDE Area Day Type

The day types associated with the PLANTWIDE area are:

Day Type ID	Purpose
PRODUCTION_- DAY	Used for days in which production is run
MAINT_ONLY_- DAY	Used for days when only maintenance crew

KILN_ONLY and ASSEMBLY Area Day Type

The day types associated with both the KILN_ONLY and ASSEMBLY areas is:

Day Type ID	Purpose
PRODUCTION_- DAY	Used for days in which production is run

Sample Factory Schedule Configuration

There are different base schedules for each area.

PLANTWIDE Area Base Schedule

The base schedule associated with the PLANTWIDE area is:

Day Type ID	Event ID	Time	Offset Flag
PRODUCTION_DAY	LIGHTS_ON_EV	5:30	
PRODUCTION DAY	START BREAK EV	10:30	
PRODUCTION DAY	END BREAK EV	10:45	0
PRODUCTION DAY	START LUNCH EV	12:00	
PRODUCTION_DAY	END_LUNCH_EV	12:45	0
PRODUCTION_DAY	START_BREAK_EV	14:15	
PRODUCTION DAY	END BREAK EV	14:30	0
PRODUCTION_DAY	START_LUNCH_EV	17:30	
PRODUCTION DAY	END LUNCH EV	18:15	0
PRODUCTION_DAY	START_BREAK_EV	20:00	
PRODUCTION DAY	END BREAK EV	20:15	0
PRODUCTION DAY	LIGHTS OFF EV	23:30	
MAINT_ONLY_DAY	LIGHTS_ON_EV	5:30	
MAINT_ONLY_DAY	LIGHTS_OFF_EV	23:30	

KILN_ONLY Area Base Schedule

The base schedule associated with the KILN_ONLY area is:

Day Type ID	Event ID	Time	Offset Flag
PRODUCTION_- DAY	KILN_ON_EV	05:30	
PRODUCTION_- DAY	KILN_WARM_- EV	05:45	0
PRODUCTION_- DAY	KILN_FULL_EV	06:00	0
PRODUCTION_- DAY	KILN_COOL_EV	23:00	0
PRODUCTION_- DAY	KILN_OFF_EV	23:15	

ASSEMBLY Area Base Schedule

The base schedule associated with the ASSEMBLY area would be:

Day Type ID	Event ID	Time	Offset Flag
PRODUCTION_- DAY	CONVEYOR_ON_EV	06:00	

PRODUCTION_ DAY	CONVEYOR_OFF_ EV	15:45	
PRODUCTION_ DAY	CONVEYOR_ON_EV	16:00	
PRODUCTION_ DAY	CONVEYOR_OFF_ EV	23:00	

Sample Factory Day of the Week Assignments

Finally, the days of the week are assigned to day types for each area as follows:

<u>Area</u>	<u>Day of Week</u>	<u>Day Type ID</u>
PLANTWIDE	SUNDAY	MAINT_ONLY_DAY
PLANTWIDE	MONDAY	PRODUCTION_DAY
PLANTWIDE	TUESDAY	PRODUCTION_DAY
PLANTWIDE	WEDNESDAY	PRODUCTION_DAY
PLANTWIDE	THURSDAY	PRODUCTION_DAY
PLANTWIDE	FRIDAY	PRODUCTION_DAY
PLANTWIDE	SATURDAY	MAINT_ONLY_DAY
KILN_ONLY	MONDAY	PRODUCTION_DAY
KILN_ONLY	TUESDAY	PRODUCTION_DAY
KILN_ONLY	WEDNESDAY	PRODUCTION_DAY
KILN_ONLY	THURSDAY	PRODUCTION_DAY
KILN_ONLY	FRIDAY	PRODUCTION_DAY
ASSEMBLY	MONDAY	PRODUCTION_DAY
ASSEMBLY	TUESDAY	PRODUCTION_DAY
ASSEMBLY	WEDNESDAY	PRODUCTION_DAY
ASSEMBLY	THURSDAY	PRODUCTION_DAY
ASSEMBLY	FRIDAY	PRODUCTION_DAY

Configuring the Action Calendar

Action Calendar at a Glance

The Action Calendar gives you the capability to:

- Define and automatically execute events in your plant, based on a standard schedule.
- Project a schedule of automated events for any day in the future
- Add, delete or reschedule events for a specific day.

Once you have planned your areas and events, entering information into the Action Calendar is very straightforward.

- Open the Action Calendar
- Action Calendar window parts
- Action Calendar printing

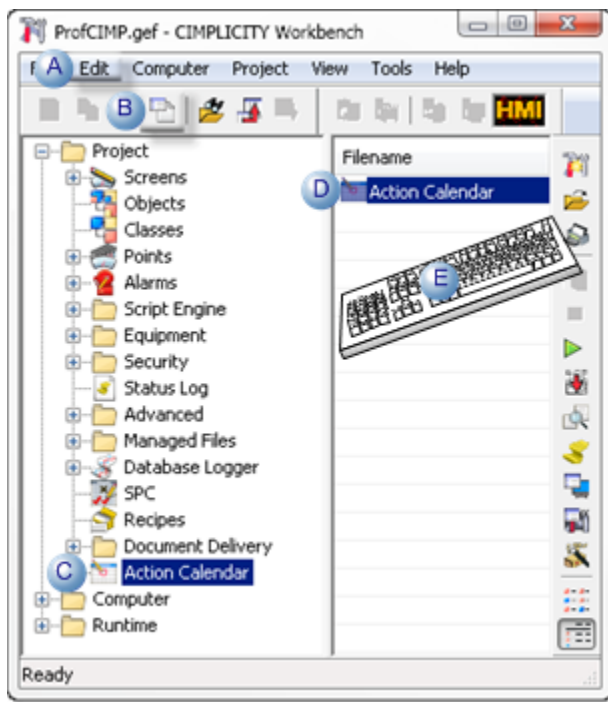
Open the Action Calendar

CIMPLICITY provides several methods to open the Action Calendar.

1. Select **Project>Action Calendar** in the Workbench left pane.

Note: If the icon does not appear in the Workbench left pane, you may need to enable the Action Calendar option in your project.

2. Select Action Calendar in the Workbench right pane.
3. Do one of the following.



A	Click Edit>Properties on the Workbench menu bar.
B	Click the Properties button on the Workbench toolbar.
C	In the Workbench left pane:


	Either	Or
	Double click Action Calendar .	a. Right-click Action Calendar . b. Select Properties on the Popup menu.
D	In the Workbench right pane:	
	Either	Or
	Double click Action Calendar.	a. Right-click Action Calendar. b. Select Properties on the Popup menu.
E	Press Alt+Enter on the keyboard.	

Action Calendar window parts

When you open the Action Calendar, you see the Action Calendar window that is divided into four parts:

- Area
- Schedule Type
- [Day Type Legend \(on page 972\)](#)
- Weekday Schedule

Area

The Area box displays the current area being viewed, for example an area called Plant. Selecting the new menu option accessed by clicking the right mouse button over the control or selecting the menu button can configure new areas .

Schedule Type

The Schedule Type group contains radio buttons for

Day Type	to configure and view the events for a type of day.
Calendar	to view and override the schedule for a particular date.

Day Type Legend

The Day Type Legend grid provides configuration and selection of the day type being viewed. Each day type is assigned a color. Each day of the week (for example, Tuesday) displays the color of the day type to which it is assigned.

When your **Schedule Type** is in Day Type mode, the Day Type Legend displays the:

- Color and Day Type grid
- Days: Mon through Sat

When your **Schedule Type** is in Calendar mode, the Day Type Legend displays the:

- Color and Day Type grid
- Year
- Month
- Days—Mon through Sat
- Days of the month: from 1 – 31 depending on how many days are in the displayed month.

Weekday Schedule

The Weekday Schedule section

- Allows the configuration and viewing of the events configured for a day type or a projected calendar schedule. You can add or remove events from the schedule.
- Tags the hour intervals for which events are scheduled. These tags remain stationary even when you scroll the schedule up and down.

When you view:

- Day Type schedules and select a day type in the Day Type Legend you will see the day type's associated schedule in the schedule area.
- Calendar schedules you will see the schedule for the date that is selected on the calendar.

Action Calendar Printing

You can print any selected Action Calendar schedule.

4. Right-click **Action Calendar**.
5. Select Properties on the Popup menu.
6. Right-click Action Calendar.
7. Select Properties on the Popup menu.
8. Click File on the Action Calendar menu bar.
9. Do any of the following.
 - Select Print Preview.

A Print Preview window opens displaying the document that will be printed.

- Select Print Setup.

A Windows Print Setup dialog box opens enabling you to set up the appropriate printer.

- Select Print.

A Windows Print dialog box opens enabling you to enter print specifications and print the Action Calendar schedule.

**Note:**

You can also click the Print button on the Action Calendar toolbar to open the Print dialog box.

Action Calendar Data Entry Overview

When you have completed planning each area's events, day types and schedules and are ready to enter data into the Action Calendar, it is recommended that you enter information in the following order.

When you have completed planning each area's events, day types and schedules and are ready to enter data into the Action Calendar, there is an efficient configuration sequence..

The sequence is:

- Configure areas.
- Create day types.
- Assign days of the week to day types.

- Create events before or while scheduling.

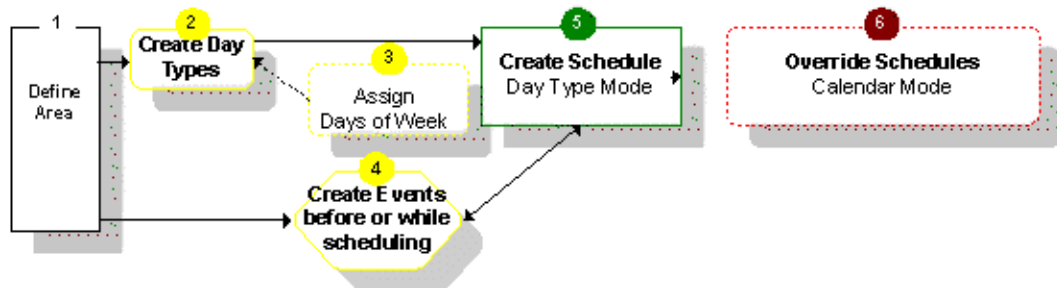
- Create a schedule using Day Type mode.
- Override schedules in Calendar mode.
- Factory Action Calendar schedule example

Factory Action Calendar Schedule Example

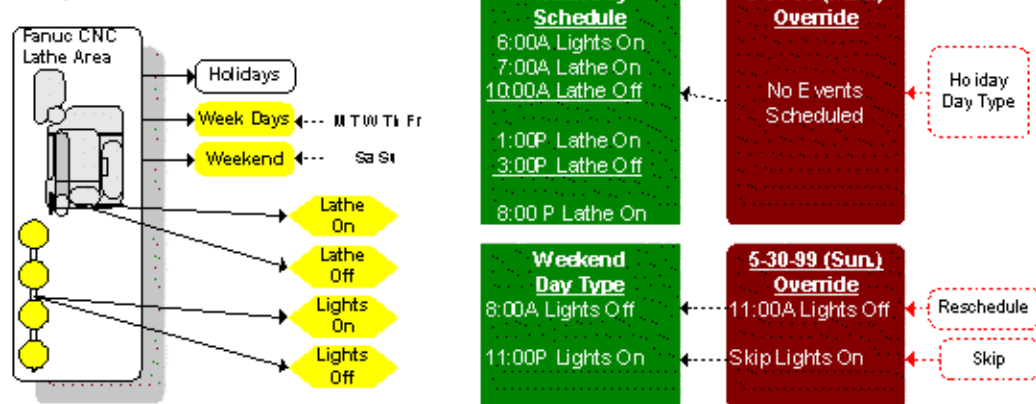
<u>Area</u>	<u>Day of Week</u>	<u>Day Type ID</u>
PLANTWIDE	SUNDAY	MAINT_ONLY_DAY
PLANTWIDE	MONDAY	PRODUCTION_DAY
PLANTWIDE	TUESDAY	PRODUCTION_DAY
PLANTWIDE	WEDNESDAY	PRODUCTION_DAY
PLANTWIDE	THURSDAY	PRODUCTION_DAY
PLANTWIDE	FRIDAY	PRODUCTION_DAY
PLANTWIDE	SATURDAY	MAINT_ONLY_DAY
KILN_ONLY	MONDAY	PRODUCTION_DAY
KILN_ONLY	TUESDAY	PRODUCTION_DAY
KILN_ONLY	WEDNESDAY	PRODUCTION_DAY
KILN_ONLY	THURSDAY	PRODUCTION_DAY
KILN_ONLY	FRIDAY	PRODUCTION_DAY
ASSEMBLY	MONDAY	PRODUCTION_DAY
ASSEMBLY	TUESDAY	PRODUCTION_DAY
ASSEMBLY	WEDNESDAY	PRODUCTION_DAY
ASSEMBLY	THURSDAY	PRODUCTION_DAY
ASSEMBLY	FRIDAY	PRODUCTION_DAY

Simple Data Entry for an Area Called Fanuc CNC Lathe

Procedure



Example



Area Configuration

Area Configuration

You will do all of your scheduling in one or more areas.

Areas can be physically independent or logical.

- Add new areas.
- Edit areas.
- Delete areas.


Add New Areas

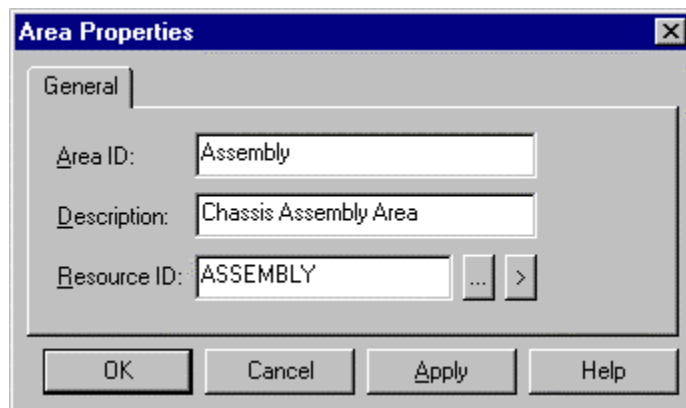
1. Do one of the following.

Method 1:

- Click File on the Action Calendar's menu bar.
- Select New.
- Select Area.

Method 2:

- Click the **Popup menu** button  to the right of the Area dialog box.
- Select New.



2. Enter the following information in the Area Properties dialog box:


Area Id	The name of the area, 15 mixed case characters or less.
---------	---

De- scrip- tion	A 40 character or less description used by you to provide more information about the area.
Re- source ID	The Resource ID to use for the area. Resources can be used to implement access control for the user interface. If you don't know yet what sort of security you will implement, select the \$SYSTEM resource. The configuration can always be changed later.

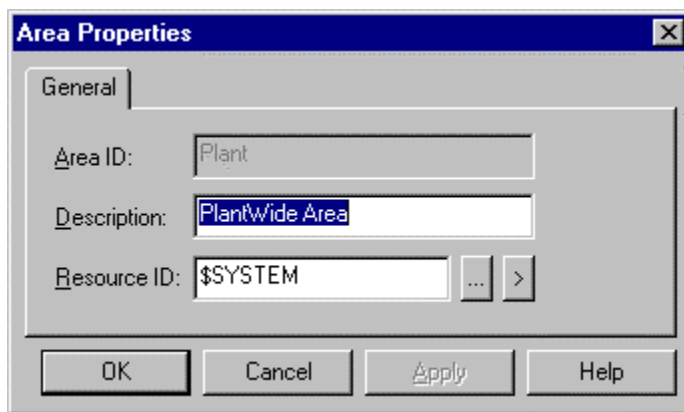
Edit Areas

1. Select the area you want to edit in the **Area** field.



2. Click the **Popup menu** button  to the right of the **Area** field...
3. Select Edit.

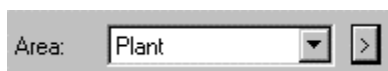
The Area Properties dialog box appears.



4. Change the **Description** or the **Resource ID**.

Delete Areas

1. Select the area in the **Area** field.



2. Click the **Popup menu** button .

3. Select **Delete** from the menu.

The area is deleted.



Important:

When an area is deleted, all of its day types, events and scheduling information are also deleted.

Day Type Legend

Day Type Legend

Before you begin to schedule events, you need to define day types and assign days of the week to the day types, for each area in your plant.

You can create day types that you:

- Use immediately and to which you assign one or more days of the week (for example, Friday).
- Reserve for future use, by creating it but not assigning any days of the week to it.

These reserved day types can then have days of the week assigned to them whenever their schedules are needed. This feature is particularly useful when one schedule replaces another for extended periods of time.



Note:

If you only need to change the schedule for a few Thursdays, you can override each Thursday that requires the long production run. You do this when the selected Schedule Type is in Calendar mode.

Example

A cutting machine, in the Cutting Area, normally runs from 1:00pm through 4:00pm every Monday through Friday.

- You schedule the Action Calendar to turn the machine on and off.
- The Cutting Area has already been defined.

Now you will create a day type in the Cutting Area called Weekdays and assign Monday through Friday to that day type.

- The plant manager tells you that every Thursday the cutting machine will have to run from Noon through 6:00pm.

You create a day type called Long Run and assign Thursday to that day type. Every Thursday the Action Calendar will run the Long Run Schedule. You do this when the Action Calendar is in Day Type mode.

Creating New Day Types

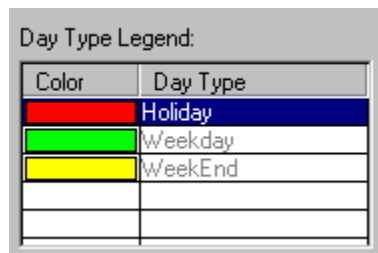
1. Do one of the following.

Method 1

- a. Click File on the Action Calendar menu bar.
- b. Select New.
- c. Select Day Type.

Method 2

Double Click on an empty row in the **Day Type** grid.

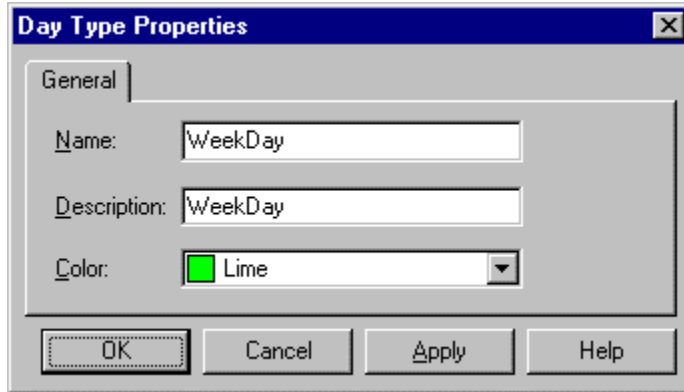


Color	Day Type
Red	Holiday
Green	Weekday
Yellow	WeekEnd

Method 3

- a. Click the right mouse button on the **Day Type** grid.
- b. Select New.

The Day Type Properties dialog box appears.



2. Enter the following information in the Day Type Properties dialog box:

Name	The 16 character mixed case name for the day type.
Description	A 40 character or less description used by you to provide more information about the day type.
Color	A color used to represent the day type graphically in the week and month calendars. Black is an invalid color, it is used to represent unassigned days.

Editing Day Types

Editing Day Types

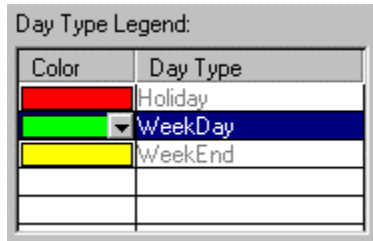
You can:


- Change the color that represents a day type.
- Edit the properties of a day type.

Procedure to Change Day Type Color

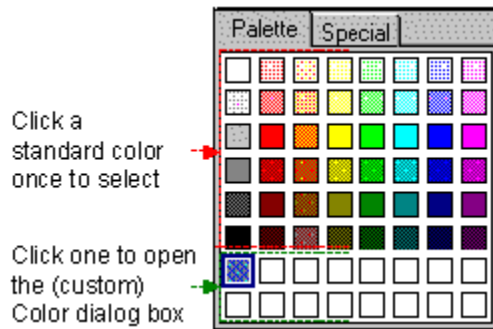
1. Click the color directly in the **DayType** grid.

A menu arrow appears.



2. Click the down arrow .

A Color palette appears.



3. Either:

- Click the standard color you want once. It will appear both in the grid and, for the corresponding days, on the Weekday Title bar.
- Click a custom color or custom color square. The Color dialog box appears in which you can create a custom color.

Procedure to Edit Day Type Properties

Do one of the following.

Method 1

Double click the entry in the **Day Type** grid.

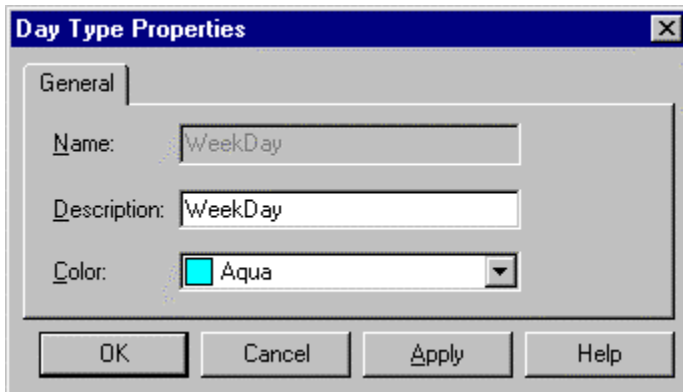
Method 2:

1. Click the right mouse button menu on the item

A menu appears.

2. Select Edit.

Using either method, the Day Type Properties dialog box appears for the Day Type that you want to edit.



Copy Day Types

You can copy an existing day type to a new day type within an area. To get the greatest benefit from this feature, use it after you have created the original day type's entire schedule. When you do, you will copy the entire schedule for the original day type to the new day type.

This feature is particularly useful when you want to create a new day type that contains a slightly modified schedule from the original. You can then easily edit the new day type and assign days of the week to it whenever the modified schedule is required.

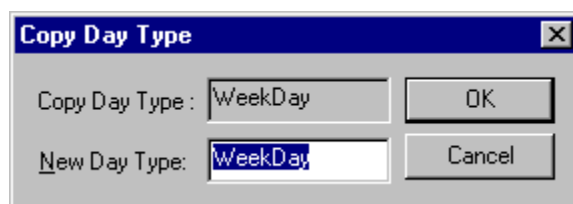


todo:

To copy a day type:

1. Select the day type in the **Day Type** grid that you want to copy.
2. Click the right mouse button.
3. Select Copy from the menu that appears.

A Copy Day Type dialog box appears.



4. Type the name of the new day type in the **New Day Type** field.
5. Click **OK**.

The Action Calendar creates a new day type with the name you specified and copies the original day type's weekday schedule to the new day type.

Example

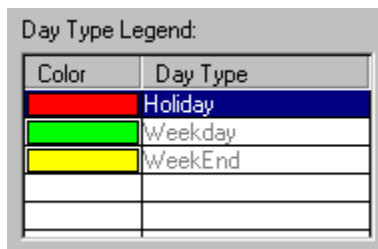
For example, you have a Weekday schedule for the spring and summer to start and stop a molding machine that is in the Molding area . However, in the fall and winter you need to lengthen the molding machine's run time on Wednesday and Thursday. You create a Long Run day type by copying the Weekday schedule and editing it to extend the molding machine's runtime hours. When the fall season begins, you can then assign Wednesday and Thursday to the Long Run day type.

Delete Day Types

Do one of the following.

Method 1:

1. Select the day type in the **Day Type** grid.
2. Click **Delete**.



Color	Day Type
	Holiday
	Weekday
	WeekEnd

Method 2:

3. Select the day type in the **Day Type** legend.
4. Click the right mouse button.
5. Select the delete option.



Note:

A day type cannot be deleted when it has a day of the week assigned to it or is used as a day type override.

Assign Days of the Week to Day Types

Once you have created day types you can assign each day of the week that requires scheduling in an area to its appropriate day type.

Example

For example, if you created a day type called Weekend and your plant adheres to a weekend schedule on both Saturday and Sunday, you assign Saturday and Sunday to the Weekend day type.



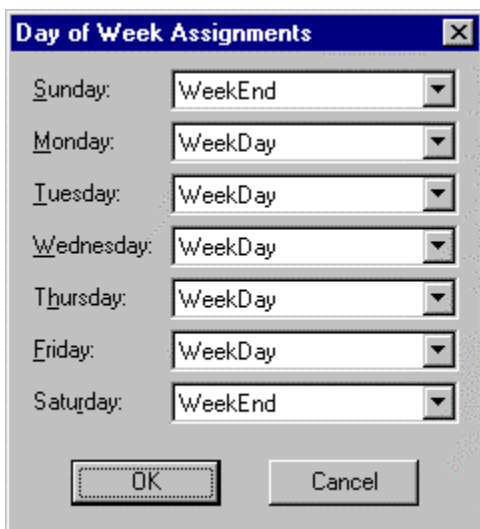
Note:

In each area, you can only assign a day of the week (for example, Monday) to one day type.

However, you can have day types to which no days of the week are assigned

When you assign a day of the week to a day type, the day of the week (for example, Wed) displays in its day type color. The Action Calendar displays days that you have not yet assigned in black. Once you assign a day of the week to a day type you can assign it to a different day type. However, you cannot change it back to being unassigned.

There are several methods to assign days of the week to day types. Following is a description of three methods to assign days of the week to day types. You perform the first two in Day Type mode, the third in Calendar mode



When you select Day Type mode in the Schedule Type group (above the **Day Type** Legend) you will see the days Sun through Sat underneath the grid. (You will not see days of the month.). While in this mode you can either call up the Day of Week Assignments dialog box or use a shortcut method..



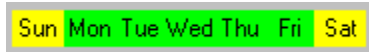
todo:

To assign a day of the week to a Day Type:

Do one of the following.

Method 1: Use the Day of Week Assignment dialog box

1. Double click the appropriate day of the week from the row of days.



A Day of Week Assignments dialog box appears.

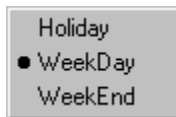
2. Select the day type from each day's menu field.

Method 2: Use a shortcut

3. Click the right mouse button over the appropriate day of the week (for example Tue) from the row of days.

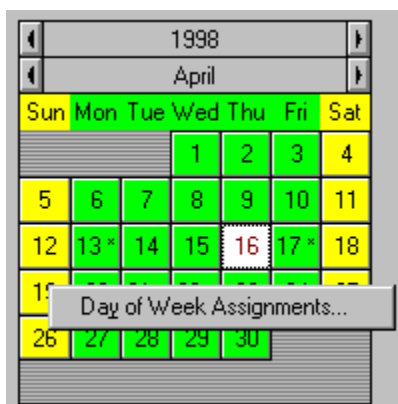
A Popup menu lists all of the day types you configured for the area.

4. Select the day type to assign to the day.



Method 3: Use the Calendar mode

5. Click the right mouse button on the **Day of the Month** grid.
6. Select the Day Of Week Assignments menu option.



A Day of Week Assignments dialog box appears.

7. Select the day type from each day's menu field.

Event Configuration

Event Configuration

In addition to creating day types for each area, you need to create the events that may be scheduled during one or more day types in that area.

You can create new events either before or during scheduling.

Working with events includes:

- Creating a new event, which can include a series of one or more actions that make up the event.
Actions include:
 - Setpoints
 - Alarm generation
 - Scripts
 - Recipes.
- Modifying an existing event
- Scheduling the event to occur at specific times in your schedule
- When necessary, changing the event. All of your changes affect all instances of the event in the schedule.
- Creating offset events to expedite scheduling

Create New Events

You can create a new event:

- Before you begin scheduling, when the Schedule Type is in Day Type mode.
- While you are entering schedules and the Schedule Type is in Calendar mode.

1. Do one of the following.

Method 1: In Day Type Mode or Calendar Mode

- a. Select File from the Action Calendar's menu bar.
- b. Select New.
- c. Select Event.


Method 2: In Day Type or Calendar Mode

- a. Select Edit from the Action Calendar's menu bar.
- b. Select Events.

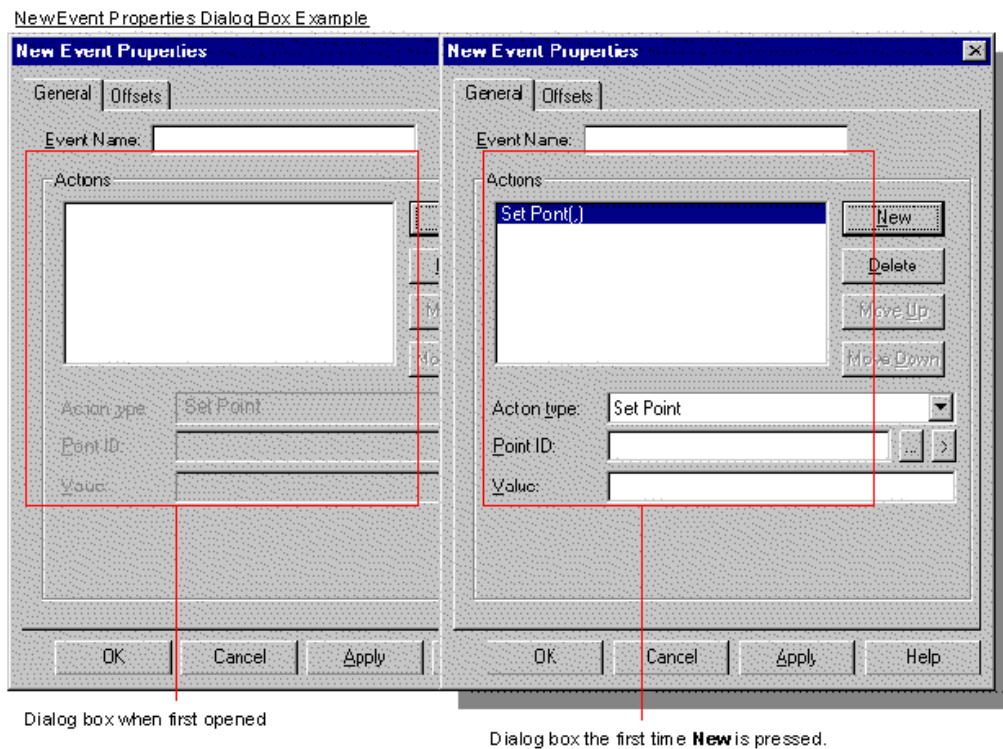
An Events dialog box displays. .

- a. Right click on any event in the Event dialog box tree.
- b. Select New.

Method 3: In Calendar Mode

- a. Select the day type for which you are creating a schedule.
- b. Select the time that the event will occur.
- c. Click the right mouse button.
- d. Select New from the popup menu.
- e. Click the Popup Menu button  to the right of the Event dialog box.
- f. Select New.

When you complete any method, the General tab of the New Event Properties dialog box displays.



2. Use the side buttons on the General tab of the New Events Properties dialog box as follows:

New	Add new actions.
Delete	Delete an existing action.
Move Up / Move Down	Position an Action item in the desired order of execution.

The fields you fill in on the bottom of the General tab of the New Events Properties dialog box depend on what action you select in the **Action type** field.

Ac- tion type	The default action is Setpoint(.). This action appears in the Actions box the first time you click New . You can configure that action or select another from the Action type. menu field. If you select another action the fields will change to reflect your choice.
---------------------	--

Modify Existing Events

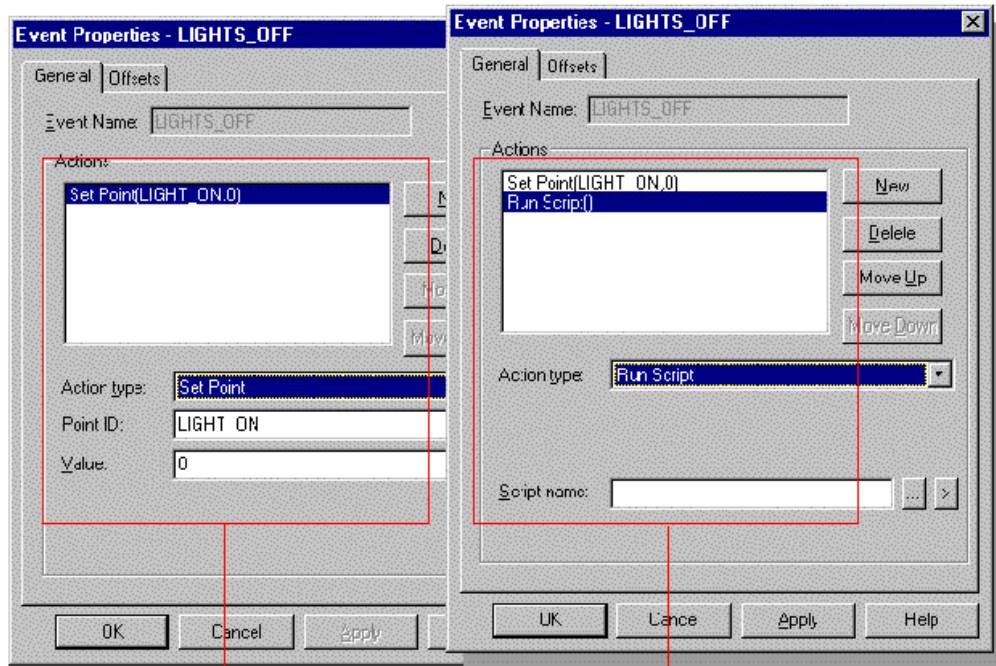
1. Click **Edit** on the Action Calendar menu bar.

The Events dialog box appears.

2. Select the event you want to modify.
3. Right click the event.
4. Select **Edit**.

Result: The General tab of the Event Properties dialog box appears, displaying the event you want to modify.

Edit Event Properties Dialog Box Example



Event Properties dialog box when opened to edit.

Event Properties dialog box when New is pressed and a new action added.

5. Use the buttons in the General tab of the Event Properties dialog box as follows:

New	Add a new action to the list.
Delete	Delete an action from the list.

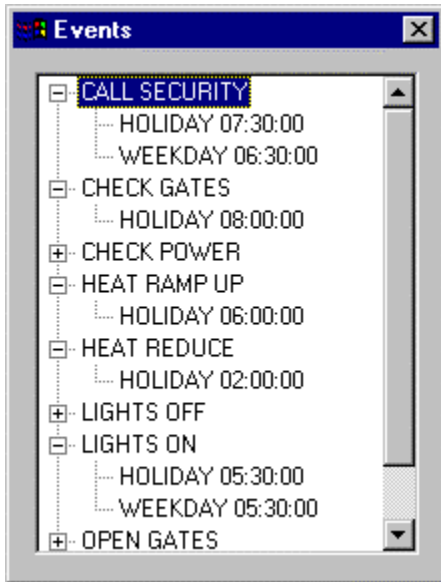
The fields you fill in on the bottom of the Event Properties dialog box depend on what action you select in the **Action type** field.

Ac- tion type	If you did not select an action when you created the event, the default action, Setpoint(), appears in the Actions box the first time you click New . You can configure that action or select another from the Action type menu field. If you select another action the fields will change to reflect your choice.
---------------------	---

View Events

1. Click Edit on the Action Calendar's menu bar.
2. Select **Events**.

The Events dialog box displays.



When you are in the Events dialog box:

- At first, you will see a tree of the events along with scheduling information for each event..
- As you navigate through the tree, the hour Schedule part of the Action dialog box will update and position the corresponding event.
- Events can be:
 - Deleted
 - Added
 - Edited
 - Scheduled
 - Unscheduled



Note:

Events can only be deleted when they are not scheduled.

Configure Offset Events

Offset Events allow you to specify a sequence of events that always occur in a certain order and at a certain time.

Offset Events provide you with a way to expedite any rescheduling that occurs for the sequenced events by reducing the number of calendar entries you have to make for each time the sequence is scheduled to one (1). Needless to say this also minimizes the possibility for error.

**Important:**

Offset events only support one level of offset. You cannot create an offset of an offset.

In the next example, the Kiln_Start event, which is an offset of Kiln_On cannot have an event that is scheduled static to when it occurs. For example, Kiln_Preheat has to occur static to Kiln_On (20 minutes prior). It cannot be configured to occur 10 minutes after Kiln_Start. Simply said, if you are a programmer—offsets cannot be nested.

Example

For the sake of the example, say your factory has a Kiln. To start this Kiln, you must perform the following events in the following order:

1. **Kiln_Start.** Turn on Kiln
2. **Kiln_Preheat.** Preheat Kiln (10 minutes later)
3. **Kiln_On_Full.** Kiln on Full (20 minutes later)

There are at least three methods to configure your schedule. However, the first is inefficient; the second is problematic; the third is the best choice. See the example below.

**Important:**

Events with offset that wrap around the end of the day are not supported.

Example

You schedule an event for 11:59 PM on March 1 and an offset event of two minutes. The offset event will be scheduled for 12:01 AM on March 1, NOT 12:01 AM on March 2.

Method 1. (Inefficient)

You could easily configure three scheduled events to occur.

4. 7:00 – Kiln_Start
5. 7:10 – Kiln_Preheat
6. 7:30 – Kiln_On_Full

However, if you decide to start the Kiln 30 minutes earlier tomorrow, you need to reschedule three events. Or if you decide that the preheat cycle can be decreased to 10 minutes, you need to move all Kiln_On_Full events back 10 minutes in all of our schedules.

Method 2. (Problematic)

One possible offset configuration is, in the Offset tab of the Event Properties dialog box, to configure the event:

7. Kiln_Start
8. Kiln_Preheat to happen 10 minutes later
9. Kiln_On_Full to be 30 minutes after Kiln_Start

This way when you schedule Kiln_Start the other two events are automatically scheduled.

The problem with this strategy is that you need the Kilns to be on at 7:30. If the preheat cycle is decreased by 10 minutes you still need to change your schedules to move Kiln_Start forward .

Method 3. (Best method)

The correct solution is illustrated below. In the Offset tab of the Event Properties dialog box, configure the event:

10. Kiln_On that has no actions.
11. Kiln_Start to occur 30 minutes prior to KILN_ON
12. Kiln_Preheat to occur 20 minutes prior to KILN_ON
13. Kiln_On_Full to happen at the same time as KILN_ON.

Now if you need the kilns to be ready at 7:30 you simply place the Kiln_On event at 7:30.

Create Offset Events

1. Open the Events dialog box.
2. Select the event that will have offset events.
3. Either:
 - a. Select Edit on the event's popup menu.
 - b. Select New to create a new event and associated offset events.

The Event Properties dialog box opens.

4. Select the [Offset \(on page 970\)](#) tab.

The Offset tab of the Event Properties dialog box appears.

5. Use the buttons in the Offset tab of the Event Properties dialog box as follows:

New	Add a new offset event to the list.
Delete	Delete an offset event from the list

6. Enter the following information in the Offset tab of the Event Properties dialog box fields:

Event	Name of an offset event.
Off-set	Time that should pass between the initial event and the offset event. If the offset event occurs before the initial event, make a negative entry; after, make a positive entry in an HH:MM:SS format.

Schedules

Schedules

When you have defined a plant's areas and configured each area's day types, you are ready to configure schedules.

Schedule

A schedule defines the series of activities that should occur on the days of the week that are assigned to a day type.

If you created the area's events, you now only need to schedule them. If you did not create the events, you can create them during scheduling.

First, when your Schedule Type is in Day Type mode, configure the basic day type schedules for each area.
You configure a schedule of events for each day type. As a result, the Action Calendar will apply a day type's schedule of events to each day of the week assigned to it
Later, you can switch your Schedule Type to Calendar mode and override any specific days or events, where necessary.

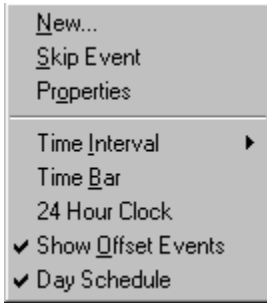
Example

You assign Monday, Wednesday and Friday to a day type called Weekday. The Action Calendar will carry out whatever schedule you create for the day type Weekday on Monday, Wednesday and Friday.

Before you get started, you may want to take a look at how you can adjust the Schedule View.

Adjust the Schedule View

There are several options available to you when viewing the schedule. The options can be found on the applications view menu, or by using the right mouse button on the schedule.




Option	Description
Time Interval	Allows you to specify the time interval to use in the day view. Valid intervals are 10, 15, 30 and 60 minutes.
Time Bar	Toggles the time bar on the left of the schedule. The time bar allows you to rapidly find events.
24 Hour Clock	Enables you to switch back and forth between a 24 hour clock and a 12 hour AM/PM clock.
Show Offset Events	Lets you decide if offset events are viewed in the schedule.
Day Schedule	Lets you toggle back and forth between a day view and a list to display the scheduled events. If you find the day view becoming cluttered due to many events scheduled in the same time slot, try the list view.

Add/Modify/Delete a Scheduled Event in Day Type Mode


- Add a scheduled event.
- Modify a scheduled event.
- Delete a scheduled event.

Add a Scheduled Event

If the event already exists:

1. Fill in the time for the event to occur.
2. Click the **Browse** button  to the right of the **Event** field.
3. Select the appropriate event to schedule.

If the event does not exist:

- a. Click the **Popup menu** button  to the right of the **Browse** button.
- b. Select New from the popup menu.
- c. [Configure a new event. \(on page 975\)](#)



guide:

Guideline: Scheduling Time

Actions cannot be scheduled to at the exact same time as the start of the day. This is because this is the transition period from one day schedule to the next, and this time is ambiguous.

To help work with this limitation the start of day configuration has been modified so that the start of the day can be configured with 1 minute resolution.

The start of the day should be selected to coincide with a time between shifts that

- Will have no need for scheduled activities
- Is a natural breaking point from 1 day to the next.

Midnight is often a good time for this, but for others 3:00 am may be better.

Modify a Scheduled Event

4. Make sure the Action Calendar is in Day Type mode.
5. Double click on a scheduled event in an hour row of the Action Calendar Schedule part.

The Scheduled Event dialog box opens.

6. Do one of the following:
 - Change the time of the event.
 - Change the event.
 - Select Edit from the **Event** field popup menu to open the Event Properties dialog box.

Delete a Scheduled Event

7. Select the scheduled event in the hour Schedule part of the Action Dialog box.
8. Press the **Delete** key.

Configuring Schedule Overrides

Configure Schedule Overrides

When you have completed configuring an area's basic schedule, you may have specific days or events that will need to be changed during a specified calendar day.

You can make these changes through:

- Day type overrides.
- Event overrides.

Day Type Overrides

Day Type Overrides

Day Type Overrides allow you to change the day type for a specified calendar day. You might need to make the change for a variety of reasons including holidays, long weekends, plant shutdowns or to accommodate a longer production schedule.

Example

You normally run your cutting machine, in the Cutting Area, from 1:00pm through 4:00pm every Monday through Friday. You schedule the Action Calendar to turn the machine on and off. You have all ready defined the Cutting Area. Now you will create a day type in the Cutting Area called Weekdays and assign Monday through Friday to that day type.

The plant manager tells you that beginning three weeks from the current week, the cutting machine will have to run from noon through 6:00pm for four Thursdays in a row.

Anticipating these long runs you have created a day type called Long Run that turns the machine on at noon and off at 6:00pm. You can immediately select the four involved Thursdays and override the Weekdays day type with Long Run. On those four Thursdays the Action Calendar will follow the Long Run schedule.

**Note:**

: Switch the Schedule Type to Calendar mode to assign Day Type Overrides.

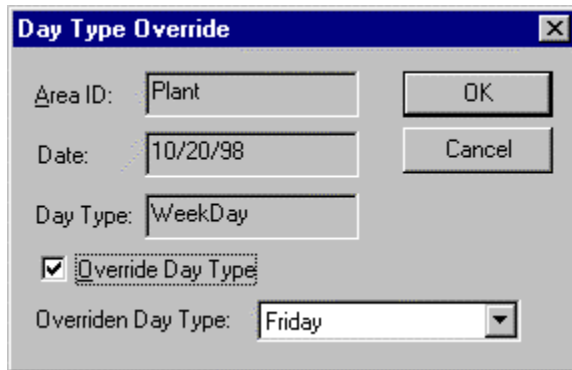
You can do the following with Day Type Overrides:

- Add
- Remove
- View
- Edit
- Delete

Add a Day Type Override

1. Double click on a day in the Month calendar.

The Day Type Override dialog box appears.



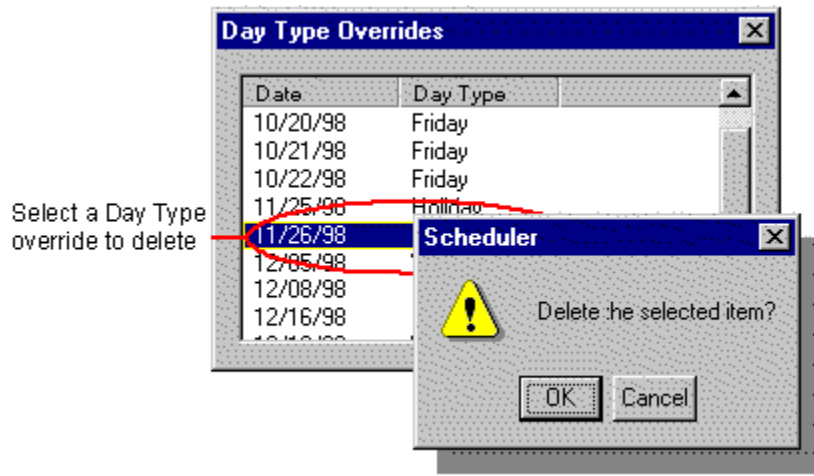
The Day Type Override dialog box tells you (read only) the:

- Area you are in
 - Date you selected
 - Currently assigned day type
2. Enter the following information in the Day Type Override dialog box to override the day type.

Override Day Type	Select to override the currently assigned day type
New Day Type	The day type to use instead of the currently assigned day type

Remove a Day Type Override

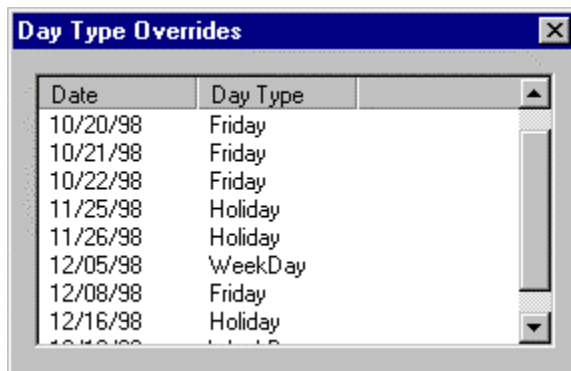
1. Double-click on the day.
2. Uncheck the **Override Day Type** check box in the Day Type Override dialog box.



View Day Type Overrides

1. Select the application's Edit menu.
2. Select Day Type Overrides.

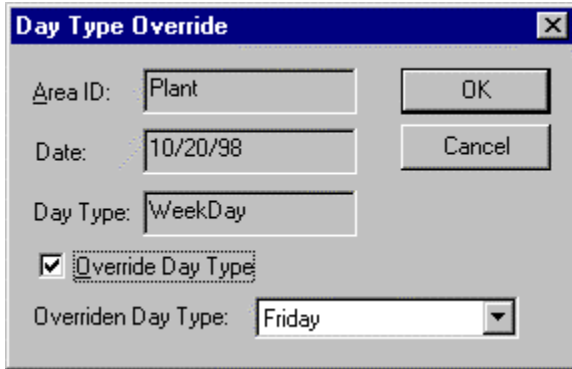
The box opens.



Edit a Day Type Override

Double click the entry.

The Day Type Override dialog box appears.

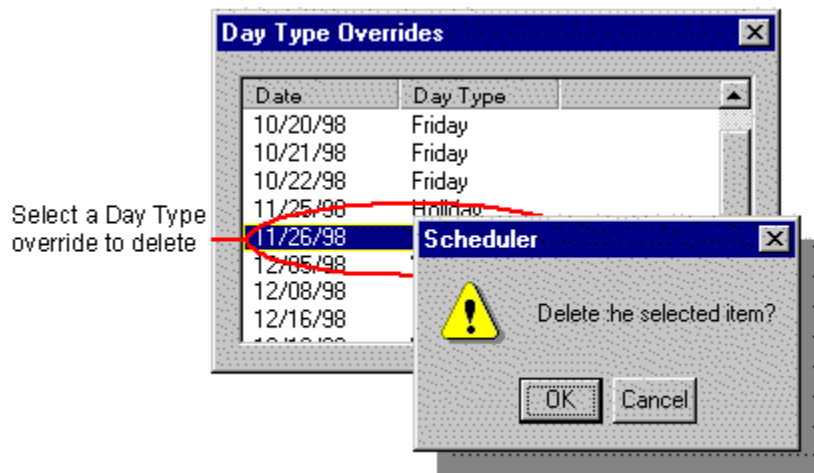


Delete a Day Type Override

1. Select the entry to be deleted.
2. Press **Delete** on the keyboard.

A message appears to confirm deletion.

3. Click **OK**.



Event Overrides

Event Overrides

Event Overrides allow you to change an event in a schedule during a specific calendar day.

For example, you want to start production an hour early tomorrow or extend lunch by an hour. You use an event override to make the change.

There are three types of event overrides.

- New Scheduled Event Override.
- Rescheduled Event.
- Skip a Scheduled Event.

You can do the follow with Event Overrides:

- View Event overrides for a specific date.
- View all Event overrides.
- Edit
- Delete



Note:

All event overrides are performed when the Action Calendar is in Calendar mode.

Add a Scheduled Event Override

1. Delete the event you want to replace.
2. Add an [event that exists or a new event \(on page 975\)](#) to the same time.

Reschedule or skip an Event

1. Double click the event to be modified.

The Scheduled Event dialog box displays:

Scheduled Event

General

Base Event

Time : 03:30:00 AM

Event WARM UP MACHINES ... >

Event Override

Skip Event

Reschedule Event


03 : 30 : 00 AM

OK Cancel Apply Help

The Scheduled Event dialog box tells you (read only) the:

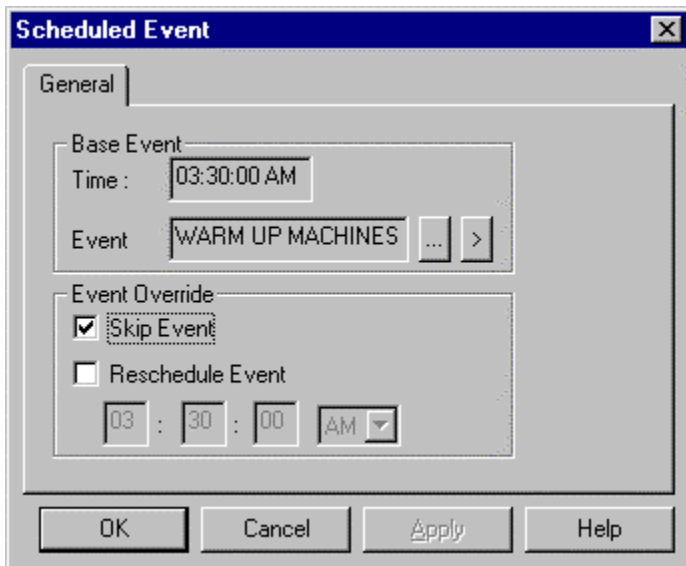
- Time the event is currently scheduled
 - Event that is currently scheduled
2. Enter the following information to skip or reschedule an event in the Scheduled Event dialog box:

Skip Event	Select to skip the event on the selected calendar day.
Reschedule Event	Select to reschedule the event on the selected calendar day.
Reschedule Time	Enter the time the event should occur on the selected calendar day. The event and all of its offsets, if any, will be rescheduled.

 **Note:**
When a day has event overrides, an asterisk is displayed next to the day in the month calendar.

Remove a Skip or Reschedule Override

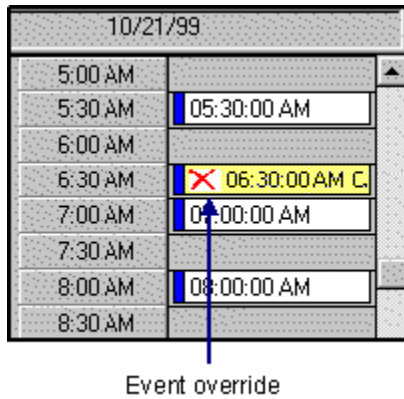
1. Double click on the override event.
2. Clear the **Skip Event** or **Reschedule Event** check box



View Event Overrides for a Specific Date

Select the date in the Day Type Legend calendar.

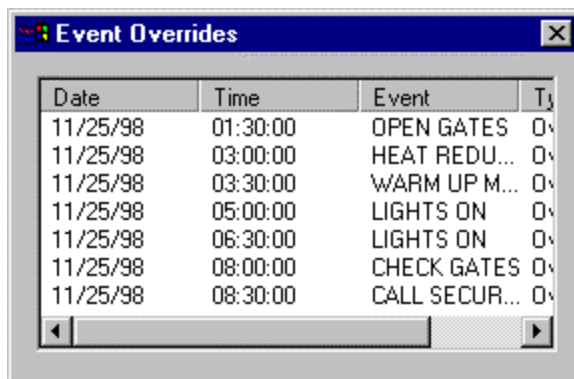
An O displays in the Schedule part of the Action Dialog box on the Events that are overridden.



View all Event Overrides Viewed

1. Select the application's Edit menu.
2. Select Event Overrides.

The Event Overrides dialog box appears.

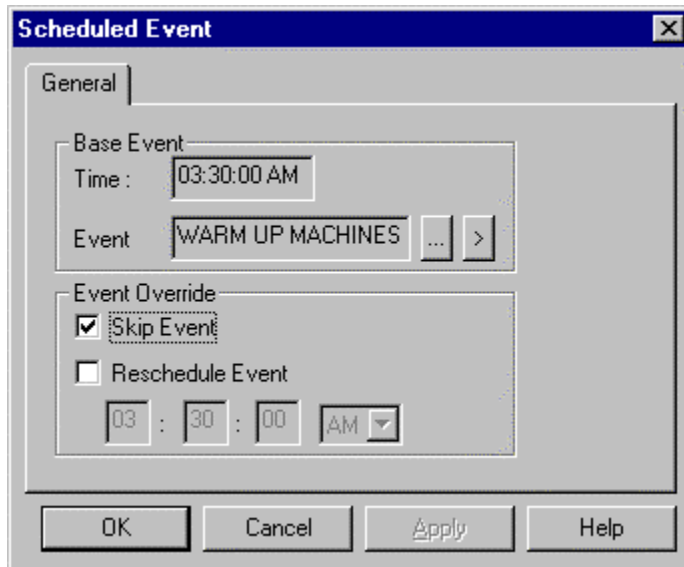


Edit Event Overrides

1. Double click on the entry.

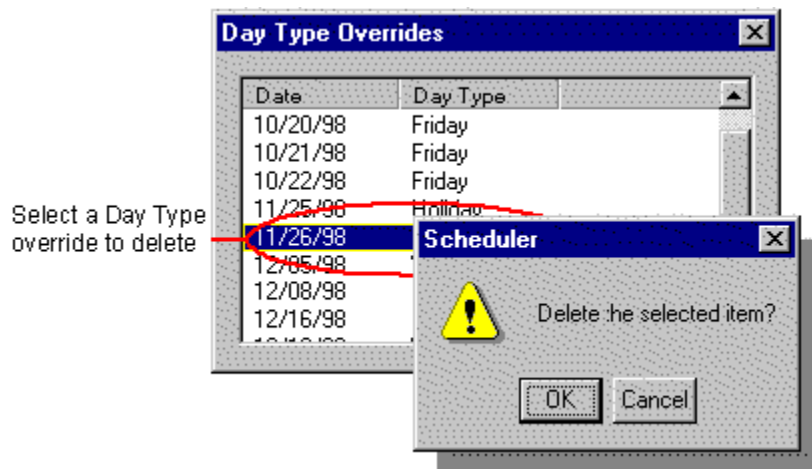
The Scheduled Event dialog box associated with that event override appears.

2. Make the required changes.



Delete Event Overrides

1. Select the entry to be deleted.
2. Press **Delete** on the keyboard.



Security

Security

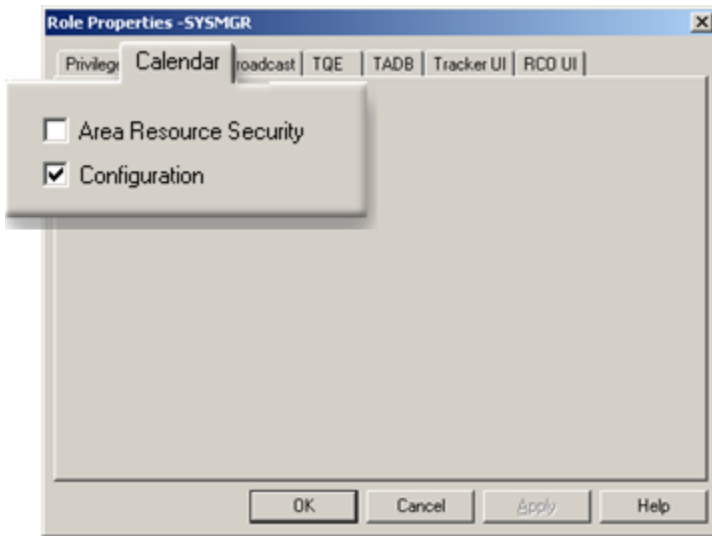
The Action Calendar provides access control to CIMPLICITY users logged into the system.

Security is provided and can be enforced only when the project is running.

When the project is not running, any user may perform configuration from the configuration cabinet.

Role Base Privileges

You, the system administrator, have assigned each CIMPLICITY user a role. Each role has a set of privileges associated with it. When the Action Calendar is part of the project, the Calendar tab of the Role Properties dialog box is displayed:



Choices for the Calendar tab of the Role Properties dialog box are:

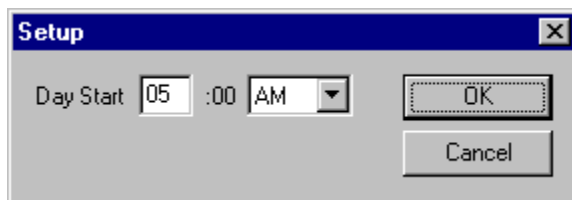
Area Based Security	When resource based security is:	
	Enabled	A user will only be able to see areas whose Resource ID is assigned to the user
	Disabled	A user will be able to see all areas.
	Example If you have schedules across several parts of your plant, you may wish to restrict the paint booth operator from modifying the assembly schedule. Resource based security is the way to do this.	
Configuration	When configuration is:	
	Checked	Users will be able to configure a schedule for any areas they can see.

	Unchecked	Users will be able to view schedules but no configuration is possible.
--	-----------	--

Procedure to set the Day Start Time

1. Select **Tools** on the Action Calendar's menu bar.
2. Select Setup.

The Setup dialog box displays.



3. Enter the following information in the Setup dialog box.

Day Start	Enter a number from 01 to 12
AM/PM	Choose AM or PM from the Menu field.



Important:

Day Start cannot be changed when the project is running. Changing the Day Start requires that the application be exited.

Command Line Parameters

The CalCfg.exe program takes command line options. These can be useful if you want to launch CalCfg from a CimView screen.

Command	Description
/AREA areald	Specifies the default area to select.
/ARE- ALOCK	Lock the current area, user cannot switch areas.
/ONLI- NEONLY	Application can only run when CIMPLICITY project is active. This is useful if you want to require operators to be logged in and subject to role privileges.

Command	Description
/PROJECT path	Specifies the path to the project's file (e.g. C:\Program Files\Proficy\Proficy CIMPLICITY\projects\cimpdemo\cimpdemo.gef) to use.
/READONLY	Do not allow any configuration to be performed while the project is or is not running regardless of Role configuration.
/TODAY	Default to viewing the schedule for the current day.

Chapter 7. Python Scripting

Overview

You can run a Python script in response to an event.



Note:

If you install Python after you install CIMPLICITY or if the *.py file association changes for any reason, the scripts that are running will not recognize the CIMP libraries because of the change in context. As a workaround, you can point the file association to **[CIMPLICITY INSTALL]\exe \proficy_code_launcher.exe**.

Steps to execute a Python script:

1. In the Event Editor, create the event for which you want to run a python script. Refer [Step 3. Configure an Event \(on page 41\)](#)
2. Create an action and add the Run Script action type to the event. Refer [Step 4. Create an Action \(on page 60\)](#) and [Run Script Actions \(on page 66\)](#)
3. Create a new Python Script or add an existing Python script to the action.



Note:

You can also create Python scripts from the **Scripts** section in the Workbench, and then add it to an event.

Video Links

Watch the following videos for demos on Python Scripting:

Basic Python Scripting - Part 1 (4:27)

<https://www.youtube.com/embed/XN1mJVKSZ7I>

Basic Python Scripting - Part 2 (5:52)

<https://www.youtube.com/embed/NEYlwoLVnfl>

Basic Python Scripting - Part 3 (6:12)

<https://www.youtube.com/embed/ka7iPTyAqo8>

When the event (for which you have configured Python script) occurs, the Python script gets executed, and you can view the status of the action in the Basic Control Engine User Interface (BCE UI). You can also stop a script execution from BCE UI. Refer [Control Scripts \(on page 936\)](#).

The Proficy Code Editor enables you to create and edit Python scripts. You can also use other editor such as VSCode with CIMPLICITY. The Proficy Code Editor files are located **<install location>/Proficy CIMPLICITY/ProficyCode**

You can access the Python distribution at **<install location>/Proficy CIMPLICITY/python3**.

proficy_code_launcher is located at **<install location>/ Proficy CIMPLICITY/exe**. It sets the environment variables to run Python. You can launch the code editor using proficy_code_launcher and set the environment variables silently.

Python APIs

You can find the Python APIs at the following location:

[oxy_ex-1/topics/cimlicity.html](#)

Additional Packages

The following additional packages are included with CIMPLICITY.

Package	Version
certifi	2021.10.8
charset-normalizer	2.0.7
colorama	0.4.4
Cython	0.29.24
idna	3.3
isort	5.10.1
lazy-object-proxy	1.7.1
mccabe	0.6.1
numpy	1.21.2
paho-mqtt	1.6.0

Package	Version
pandas	1.3.4
pip	21.3
platformdirs	2.4.1
pyodbc	4.0.32
python-dateutil	2.8.2
pytz	2021.3
pywin32	302
requests	2.26.0
setuptools	57.4.0
six	1.16.0
toml	0.10.2
urllib3	1.26.7
wheel	0.37.0
wrapt	1.13.3

You can get the list of additional packages using the command:

```
"%CIM_PYTHON_HOME%\Scripts\pip3" list --verbose
```

```
"%CIM_PYTHON_HOME%\Scripts\pip3" list --verbose | findstr /i /c:"%CIM_PYTHON_HOME%"
```

Install Third Party Packages

A Python package is the collection of Python modules, in which similar modules are grouped a separate directory.

Some packages provide source distributions (sdist) and some provide binary distributions (wheels).

When you install, pip will typically prefer a binary package if it is available.

If a binary package is not available, the source package is installed and built.

You can force using a binary package with the options

- `--prefer-binary` (prefer older binary packages over newer source packages)
- `--only-binary <format_control>` (fail if binary package not available).

You can force using a source package with the option `--no-binary <format_control>`.

Building source distributions

The following are the pre-requisites for most Python packages:

- A C/C++ development environment
- The required LIB and INCLUDE directories for any dependencies must specified in the environment.
- Visual Studio. Python will identify the files if the Visual Studio is installed in the default location.

CIMPLICITY and CIMPLICITY Python are compiled as 32-bit programs. Any dependent binary libraries must be built as 32-bit. The install documentation of the package would include the instructions for building it.

You will need to update the INCLUDE and LIB environment variables to point to any dependencies before you build. For example:

```
C:\> set LIB=C:\OpenSSL-win32\lib;%LIB%
```

```
C:\> set INCLUDE=C:\OpenSSL-win32\include;%INCLUDE%
```

Refer to the following links for more information:

<https://cryptography.io/en/latest/installation/#building-cryptography-on-windows>

<https://packaging.python.org/guides/installing-scientific-packages/>

<https://packaging.python.org/guides/installing-scientific-packages/#windows-installers>

Example: Installing Pillow package from source

The Pillow package provides image processing capabilities and is available as a binary wheel for most Operating Systems. However, if you wanted to build it from source, you could run the command:

```
pip3 install --no-binary :all: pillow
```

If you get an error you can perform the following steps:

1. Install the zlib and libjpeg libraries using vcpkg (<https://github.com/microsoft/vcpkg>) using the vcpkg command.

```
\vcpkg.exe install --triplet x86-windows zlib ijg-libjpeg
```

2. Add the vcpgk include and lib directories to the environment:

```
set LIB=%LIB%;V:\vcpkg\installed\x86-windows\lib
```

```
set INCLUDE=%INCLUDE%;V:\vcpkg\installed\x86-windows\include
```

3. Run the pip install command again.

```
>>> from PIL import Image
>>> im = Image.open('c:\\temp\\fig.png')
>>> print(im.format, im.size, im.mode) PNG (640, 480) RGBA
```

Refer to the following links for the downloads or more information on installation:

<https://pillow.readthedocs.io/>

<https://pillow.readthedocs.io/en/stable/installation.html#building-from-source>

<https://github.com/microsoft/vcpkg>

Example: Installing matplotlib package from source

The matplotlib package has binary wheels. Hence, you need not build it from source.

If you cannot download **freetype-2.6.1.tar.gz** from the Online links, you can perform the steps documented at <https://matplotlib.org/stable/users/installing/index.html#installing-from-source>

1. Get the repository from git and perform pip install from the local code. You can then download **freetype-2.6.1.tar.gz** manually and put in the **build/** directory.
2. Run the following python script:



Note:

Python distribution does not include the tkinter package. You cannot use the default backend for matplotlib. Instead, you can use the agg backend that can write to the files instead of displaying it on the screen.

```
import matplotlib
import matplotlib.pyplot as plt
```

```
import subprocess

matplotlib.use('agg')

plt.plot([1, 2, 3, 4], [1, 4, 2, 3])

plt.savefig('c:\\temp\\fig.png', dpi=100)

subprocess.run(['cmd.exe', '/c', 'c:\\temp\\fig.png'])
```

Unix-only packages

Not all packages are available on Windows.

For example, pip3 install uwsgi gives the following error message:

AttributeError: module 'os' has no attribute 'uname'.

The os.uname() function is available only in unix-based Operating Systems.

Python Scripts for Event Manager Actions

The Event Manager defines **events** that can occur. Each event can be associated with one or more **actions** that run when the event is triggered. Similarly, each action can be associated with one or more events. That association is called an **event-action**.

When an event is triggered, the associated event-actions are queued to run. We call this queued event-action an **event-action instance**. Conceptually, the event-action instance begins when the event triggers and it is complete when the action finishes running.

If the action is a Python script, the script defines an EventHandlerState class. A new EventHandlerState class object instance is created for each event-action that is defined. When an event-action instance is run, the do_event_action() method is run. If the same event-action triggers again, the same EventHandlerState object instance is used and the do_event_action() method is run again.

- If the same action is used in a different event, that is a different event-action and a different EventHandlerState class object instance is created for that event-action.
- If the same script is used in a different action, that is a different event-action and a different EventHandlerState class object instance is created for that event-action.

EventHandlerState lifecycle:

- The first time an instance of this event-action is queued to run, the script is loaded as a module if it hasn't already been loaded.
- An EventHandlerState object instance is created for this event-action. The object instance will be used for all the event-action instances that run.
- The `__init__()` method is called when the object is created.
- The `do_event_action()` method is run repeatedly for each event-action instance. The `CimEMEvent` parameter has information about the event that triggered this event-action instance.
- When the Event Manager is shutdown or when it reloads the script, the `do_shutdown()` method is run and then the EventHandlerState object instance reference is released and the script/module is unloaded.

The script is loaded as a module. This has the following implications:

- The script is loaded once, when the first event-action instance that uses it is performed.
- Each EventHandlerState class is in its own module.
- You can define module level variables in the script, outside of the class. All event-actions that use this script share those variables.
- One script can import another script and access its variables and functions using the module name prefix.

The EventHandlerState class has the following methods that you can implement:

```
def __init__(self, event_action_context: cimplicity.EMEventActionContext)
```

This method is called by Python when the class object instance is created. The `EMEventActionContext` parameter has information such as the event ID and the action ID of the event-action of the object instance. If the event-action was defined as part of a `CIMPLICITY` class, the object ID and attributes are also available.

In this method, you can do any initialization that should happen for each defined event-action.

```
def do_event_action(self, event_data: cimplicity.CimEMEvent)
```

This method is called by the Event Manager when an event-action instance is run. The `Cim` parameter has information about the event that triggered the action. There are also special members for the different event types: alarm events, point events, or the shutdown event.

```
def do_shutdown(self, event_data: cimplicity.CimEMEvent)
```

This method is called by the Event Manager when the Event Manager is shutting down or is reloading the scripts. (Script reload happens for changed scripts when you press the Update button in the Event Editor.)

In this method you can, for example, do any cleanup for things that you did in the `__init__` method.

Writing Standalone Python Scripts

Python can be used to write standalone scripts that you can run from the console.

Starting Proficy Code

To start the Proficy Code editor with the CIMPLICITY Python environment properly set up, you can run the following command from the Windows search box:

```
%cimpath%\proficy_code_launcher.exe
```

Proficy Code appears with no CIMPLICITY project context. This means, for example, you must fully qualify point IDs with the project name and any `log_status` messages will show up in the system log, not in a project log file.


Example:

You can try the following simple script:

1. Enter the following script into Proficy Code and save the file with a ".py" extension.

```
print("hello")
```

2. Run the script from the command palette by typing `Ctrl+Shift+R`, and then start typing **Python:**

Run Python File in Terminal. You can also use the play button  in the upper right corner of the window.

3. Add a simple CIMPLICITY API call. IntelliSense tool appears when you type "cimplicity." in the last statements.

```
import cimplicity
import os
print("hello")
cimplicity.log_status(cimplicity.StatusSeverity.SUCCESS,
                    os.path.basename(__file__), "hello")
cimplicity.terminate_api_in_thread()
```

4. Run the code.

Result: The log message is displayed in the CIMPLICITY system status log.



Note:

When you use the CIMPLICITY API in a thread other than the one Event Manager calls the `do_event_action` method in, you must call `cimplicity.terminate_api_in_thread` before the thread exits. (It may not be strictly necessary when only calling `cimplicity.log_status`.)

More involved script

In the following script we get and set point values.

Notes

- In the beginning of the script we use the `point_get` and `point_set` methods which are useful for quick access to points one time. If you are going to get and/or set a point value multiple times, it is more efficient to use a Point object.
- You must fully qualify the points as this CIMPLICITY environment is not associated with any project.
- Do not combine fully qualified and unqualified point references in the same client.
- A qualified point ID looks like `\\proj\pointid`. When entering that in a Python string you must escape the backslashes: `\\\\proj\\pointid`.
- When you run the script for the first time, CIMPLICITY will prompt you to log into the project.
- Call `cimplicity.terminate_api_in_thread` in a finally block to ensure it is always called.

```
import cimplicity
import os
import time

try:
    print("hello")
    cimplicity.log_status(cimplicity.StatusSeverity.SUCCESS,
                        os.path.basename(__file__), "hello")

    PTVAl = "\\SITEA\\AirConditioner_1.Humidity"
    PTVAl_SETPT = "\\SITEA\\AirConditioner_1.HumiditySetpoint"

    val = cimplicity.point_get(PTVAl)
    print(f"Current value = {val:.2f}")
```

```

val = cimplicity.point_get(PVAL_SETPT)
print(f"Current value_setpoint = {val:.2f}")

newval = input("-> Enter new value_setpoint: ")
cimplicity.point_set(PVAL_SETPT, newval)
val = cimplicity.point_get(PVAL_SETPT)
print(f"Current value_setpoint = {val:.2f}")

with cimplicity.Point() as pt:
    pt: cimplicity.Point # this declaration helps IntelliSense
    pt.id = PVAL
    for i in range(0, 10): # range(inclusive, exclusive)
        if i > 0:
            time.sleep(1) # seconds
            print(f"... value = {pt.get_value():.2f}"
                  f" (at {pt.timestamp_local})")
finally:
    cimplicity.terminate_api_in_thread()

```

Starting Proficy Code in a Project Context

If you want to work on standalone scripts for a particular project:

1. Open the project in the Workbench
2. Select the Tools > Command Prompt menu item.
3. In the command prompt, enter the following command to launch Proficy Code and open the scripts folder. `%cimpath%\proficy_code_launcher.exe scripts`



Note:

You are recommended to put your standalone scripts in a subdirectory of the scripts directory to keep them separate.



Note:

In the command prompt you can also type "python" and it will run in the CIMPLICITY Python environment.

You can follow the above steps to build up a script, but this time you will not need to use fully qualified point IDs.

```

import cimplicity

import sys

import time

class EventHandlerState:

    def __init__(self, event_action_context: cimplicity.EMEventActionContext):

        # store the object attributes for later use

        self.obj_attrs = event_action_context.object_attributes

        print(f"__init__: obj_attrs: {self.obj_attrs}")

        sys.stdout.flush()

    def do_event_action(self, event_data: cimplicity.CimEMEvent):

        cimplicity.log_status(

            cimplicity.StatusSeverity.SUCCESS, "myscript", "running")

        print(f"event_id: {event_data.event_id}")

        print(f"action_id: {event_data.action_id}")

        print(f"object_id: {event_data.object_id}")

        print(f"timestamp_local: {event_data.timestamp_local}")

        print(f"event type: {event_data.type}")

        print(f"obj_attrs: {self.obj_attrs}")

        if event_data.point_event is not None:

            print("point event:")

            pe: cimplicity.CimEMPointEvent = event_data.point_event

            print(f" point ID: {pe.id}")

            print(f" state: {pe.state}")

            print(f" quality: {pe.quality}")

            print(f" timestamp_local: {pe.timestamp_local}")

            sys.stdout.flush()

    def do_shutdown(self, event_data: cimplicity.CimEMEvent):

        pass

def do_test():

    # construct an EventHandlerState object

    ea_ctx = cimplicity.EMEventActionContext(

        "WORKUNIT03.OfflineForMaintEvent", "WORKUNIT03.OfflineForMaintAction",

```

```

    "WORKUNIT03", {"A_HASSCANNER": "0", "A_HASBUFFER": "1"})

eh_state = EventHandlerState(ea_ctx)

# construct the CimEMEvent and CimEMPointEvent objects

ts_cimp = time.time_ns() / 100

quality = (simplicity.QualityFlags.IS_AVAILABLE
           | simplicity.QualityFlags.IS_IN_RANGE
           | simplicity.QualityFlags.ALARMS_ENABLED
           | simplicity.QualityFlags.ACK_OCCURRED)

pt_event = simplicity.CimEMPointEvent(
    "WORKUNIT03.OfflineForMaintPoint", "value", quality, ts_cimp,
    simplicity.PointState.NORMAL, 0)

# call the do_event_action method

eh_state.do_event_action(simplicity.CimEMEvent(
    simplicity.EventType.POINT_CHANGE, ea_ctx.object_id,
    ea_ctx.event_id, ea_ctx.action_id, ts_cimp, None, pt_event,
    None))

if __name__ == "__main__":
    do_test()

```

Writing Common Code for Python Scripts









You can share common functions/code between your event action scripts. To do this, you can create a Python package and import it. See the [Python documentation](#) for information on how to create a package and where to locate it.

For small projects you can add the scripts to a subdirectory and import them into your script.

Example:

1. Create a subdirectory called `Common` under the project `Scripts` directory.
2. Add your common functions to a file called `helpers.py`.
3. Import the helper functions in you event action script by running `import common.helpers`.

The directory structure for the above example:

Scripts directory:	Scripts\common directory:
 <code>_pycache_</code>  <code>common</code>  <code>em_init.bcl</code>  <code>em_term.bcl</code>  <code>GenerateWebHMIModel.cs.pscript</code>  <code>Script1.py</code>	 <code>_pycache_</code>  <code>helpers.py</code>

Contents of `common\helpers.py` script:

```
import cimplicity
import datetime as dt

def fmt_response(msg: str) -> str:
    resp = f"{msg} {dt.datetime.now().isoformat()}"
    return resp

def send_response(resp: str) -> None:
    print(resp)
    cimplicity.log_status(cimplicity.StatusSeverity.SUCCESS, "event script", resp)
    cimplicity.point_set("response1", resp)
```

Contents of event action script `Script1.py`

```
import cimplicity
import common.helpers

class EventHandlerState:

    def __init__(self, event_action_context: cimplicity.EMEventActionContext):
        pass

    def do_event_action(self, event_data: cimplicity.CimEMEvent):
        resp = common.helpers.fmt_response("script ran at")
        common.helpers.send_response(resp)
        pass
```

```

def do_shutdown(self, event_data: cimplicity.CimEMEvent):
    pass

def do_test():
    EventHandlerState(None).do_event_action(None)
    pass

if __name__ == "__main__":
    do_test()

```

Testing Python Event Manager Scripts

When you create Python scripts for Event Manager, you can test the scripts from the Proficy Code Editor.

The Event Manager creates an object from your EventHandlerState class the first time an event-action attempts to run your script. Thereafter, it calls the do_event_action method to run the script. In your test code, you need to emulate these two steps.

Example:

If your script doesn't depend on the event-action context or the event context, then you can actually write you test code as follows. Note the use of `sys.stdout.flush()` to flush the output. This is necessary so that the print output shows up in the MAC_EMRP.out file.

```

import cimplicity

class EventHandlerState:

    def __init__(self, event_action_context: cimplicity.EMEventActionContext):
        pass


    def do_shutdown(self, event_data: cimplicity.CimEMEvent):
        pass

    def do_event_action(self, event_data: cimplicity.CimEMEvent):
        cimplicity.log_status(
            cimplicity.StatusSeverity.SUCCESS, "myscript", "it ran")
        print("it ran")
        sys.stdout.flush()

```

```
def do_test():
    handler_state = EventHandlerState(None)
    handler_state.do_event_action(None)

if __name__ == "__main__":
    do_test()
```

To test the script run the command from the command palette (Ctrl+Shift+P) and then select **Python: Run Python File in Terminal** or you can click  in the upper right of the window.

Advanced Script

In more advanced cases, you will want to pass a real event-action context and a real event context. Consult the CIMPLICITY Python API (oxy_ex-1/topics/cimplicity.html) for information about the properties for EMEventActionContext and CimEMEvent.

The following example represents testing a point event for a CIMPLICITY class object.

```
import cimplicity
import sys
import time

class EventHandlerState:

    def __init__(self, event_action_context: cimplicity.EMEventActionContext):
        # store the object attributes for later use
        self.obj_attrs = event_action_context.object_attributes
        print(f"__init__: obj_attrs: {self.obj_attrs}")
        sys.stdout.flush()

    def do_event_action(self, event_data: cimplicity.CimEMEvent):
        cimplicity.log_status(
            cimplicity.StatusSeverity.SUCCESS, "myscript", "running")
        print(f"event_id: {event_data.event_id}")
        print(f"action_id: {event_data.action_id}")
        print(f"object_id: {event_data.object_id}")
        print(f"timestamp_local: {event_data.timestamp_local}")
        print(f"event type: {event_data.type}")
        print(f"obj_attrs: {self.obj_attrs}")
```

```

if event_data.point_event is not None:

    print("point event:")

    pe: cimplicity.CimEMPointEvent = event_data.point_event

    print(f" point ID: {pe.id}")

    print(f" state: {pe.state}")

    print(f" quality: {pe.quality}")

    print(f" timestamp_local: {pe.timestamp_local}")

    sys.stdout.flush()

def do_shutdown(self, event_data: cimplicity.CimEMEvent):

    pass

def do_test():

    # construct an EventHandlerState object

    ea_ctx = cimplicity.EMEventActionContext(

        "WORKUNIT03.OfflineForMaintEvent", "WORKUNIT03.OfflineForMaintAction",

        "WORKUNIT03", {"A_HASSCANNER": "0", "A_HASBUFFER": "1"})

    eh_state = EventHandlerState(ea_ctx)

    # construct the CimEMEvent and CimEMPointEvent objects

    ts_cimp = time.time_ns() / 100

    quality = (cimplicity.QualityFlags.IS_AVAILABLE

               | cimplicity.QualityFlags.IS_IN_RANGE

               | cimplicity.QualityFlags.ALARMS_ENABLED

               | cimplicity.QualityFlags.ACK_OCCURRED)

    pt_event = cimplicity.CimEMPointEvent(

        "WORKUNIT03.OfflineForMaintPoint", "value", quality, ts_cimp,

        cimplicity.PointState.NORMAL, 0)

    # call the do_event_action method

    eh_state.do_event_action(cimplicity.CimEMEvent(

        cimplicity.EventType.POINT_CHANGE, ea_ctx.object_id,

        ea_ctx.event_id, ea_ctx.action_id, ts_cimp, None, pt_event,

        None))

if __name__ == "__main__":

    do_test()

```