



# **Visual Build Professional**

## **User's Manual**

Copyright © 1999-2015 Kinook Software, Inc.



# Table of Contents

<b>Part I Introduction</b>	<b>1</b>
1 Overview .....	1
2 Why Visual Build? .....	1
3 New Features .....	2
Version 4 .....	2
Version 5 .....	3
Version 6 .....	4
Version 7 .....	6
Version 8 .....	9
Version 9 .....	11
4 License Agreement .....	13
5 Installation .....	14
6 Upgrading from Older Versions .....	16
Visual Build 2/3 .....	16
Visual Build Pro 4 .....	18
Visual Build Pro 5 .....	18
Visual Build Pro 6 .....	20
Visual Build Pro 7 .....	21
Visual Build Pro 8 .....	22
<b>Part II Using Visual Build</b>	<b>23</b>
1 Main Screen .....	23
2 Menu Bar .....	24
3 Toolbars .....	24
4 Panes .....	25
Actions Pane .....	25
Step Panes .....	26
Project Steps Pane.....	27
Subroutine Steps Pane.....	28
Failure Steps Pane.....	28
Global Subroutine Steps Pane.....	28
Macros Pane .....	29
Output Pane .....	30
Properties Pane .....	30
Watches Pane .....	32
Call Stack Pane .....	32
Customizing Panes .....	33
5 Layouts .....	33
6 Building Projects .....	34
Definition of Terms .....	34
Methods of Building .....	34
Build Profiles .....	35
Build Details .....	36

Build Logging .....	37
<b>7 Dialogs</b> .....	<b>38</b>
About .....	38
<b>Action Properties</b> .....	<b>38</b>
General .....	38
GUI .....	39
Component.....	40
<b>Application Options</b> .....	<b>40</b>
General .....	41
Logging .....	42
Logging (More).....	43
File Locations.....	44
Advanced.....	44
<b>Change Password</b> .....	<b>45</b>
<b>Change Edit Password</b> .....	<b>45</b>
<b>Customize</b> .....	<b>45</b>
Toolbars .....	45
Commands.....	45
Menus .....	46
Keyboard.....	46
Options .....	47
<b>Edit Password</b> .....	<b>47</b>
<b>Find/Replace</b> .....	<b>47</b>
<b>Go To Step</b> .....	<b>48</b>
<b>Insert Macro</b> .....	<b>48</b>
<b>Macro Properties</b> .....	<b>48</b>
<b>Password</b> .....	<b>49</b>
<b>Print/Print Preview</b> .....	<b>49</b>
<b>Project Properties</b> .....	<b>50</b>
Comments.....	50
Statistics .....	50
Logging .....	50
<b>Script Editor</b> .....	<b>51</b>
<b>Step Properties</b> .....	<b>52</b>
General Tab.....	53
More Tab.....	54
Dialog Buttons.....	55
<b>User Options</b> .....	<b>56</b>
General .....	57
Projects/Steps.....	57
Display .....	58
Columns .....	59
Build .....	59
Output .....	60
Colors .....	60
Print/Backups.....	60
Miscellaneous.....	61
<b>Part III Actions</b> .....	<b>62</b>
<b>1 System</b> .....	<b>63</b>
Exit .....	63
Group .....	63
Log Message .....	63

<b>Loop</b> .....	<b>64</b>
<b>Run Program</b> .....	<b>64</b>
Program Tab.....	65
Input Tab.....	66
Advanced Tab.....	66
Remote Tab .....	68
<b>Run Script</b> .....	<b>69</b>
<b>Set Macro</b> .....	<b>70</b>
<b>Subroutine Call</b> .....	<b>70</b>
<b>VisBuildPro Project</b> .....	<b>71</b>
Project Tab.....	71
Parameters Tab.....	72
Options Tab.....	72
<b>2 Compression</b> .....	<b>73</b>
<b>7-Zip</b> .....	<b>73</b>
7-Zip Tab.....	73
Options Tab.....	74
<b>Enhanced Unzip Files</b> .....	<b>74</b>
Unzip Tab.....	75
Options Tab.....	75
<b>Enhanced Zip Files</b> .....	<b>76</b>
Zip Tab .....	76
Compression Tab.....	77
Options Tab.....	77
SFX Tab .....	78
<b>UNZIP Files</b> .....	<b>78</b>
<b>ZIP Files</b> .....	<b>79</b>
ZIP Tab .....	80
Options Tab.....	81
<b>3 Embarcadero/Borland</b> .....	<b>82</b>
<b>Make Delphi / C++Builder / RAD Studio</b> .....	<b>82</b>
Project/Group Tab.....	83
Versions Tab.....	84
Properties Tab.....	84
Options Tab.....	84
<b>Make JBuilder</b> .....	<b>85</b>
<b>StarTeam</b> .....	<b>86</b>
Server Tab.....	87
Command Tab.....	87
Global Options Tab.....	88
Options Tab.....	89
<b>4 Files</b> .....	<b>89</b>
<b>Burn CD/DVD</b> .....	<b>89</b>
Burn Tab .....	90
Options Tab.....	90
<b>Copy Files</b> .....	<b>91</b>
Copy Tab.....	92
Options Tab.....	93
Errors Tab.....	94
Attributes Tab.....	94
Robocopy Tab.....	95
<b>Create Folder</b> .....	<b>96</b>
<b>Delete Files</b> .....	<b>96</b>

<b>Delete Folder</b> .....	<b>96</b>
<b>File Checksum</b> .....	<b>97</b>
Checksum Tab.....	97
Options Tab.....	97
<b>List Files</b> .....	<b>98</b>
Files Tab.....	98
Attributes Tab.....	99
<b>Process Files</b> .....	<b>99</b>
<b>Read File</b> .....	<b>100</b>
File Tab.....	100
Text Tab.....	101
<b>Read INI</b> .....	<b>101</b>
<b>Read XML</b> .....	<b>102</b>
File Tab.....	102
XPath Tab.....	103
Results Tab.....	103
<b>Rename Files</b> .....	<b>104</b>
Rename Tab.....	104
Options Tab.....	104
<b>Replace in File</b> .....	<b>105</b>
Replace Tab.....	105
Text Tab.....	106
<b>Set File Attributes</b> .....	<b>107</b>
Files Tab.....	107
Attributes Tab.....	108
<b>Sign Code</b> .....	<b>108</b>
Sign Tab.....	109
Options Tab.....	110
Steps to Automate Code Signing.....	110
<b>Transform XML Log</b> .....	<b>111</b>
Transform Tab.....	112
Filter Tab.....	113
Parameters Tab.....	114
<b>Write File</b> .....	<b>114</b>
<b>Write INI</b> .....	<b>114</b>
<b>Write XML</b> .....	<b>115</b>
File Tab.....	115
Namespaces Tab.....	116
XPath Tab.....	116
<b>5 Help Authoring</b> .....	<b>116</b>
<b>Doc-O-Matic</b> .....	<b>116</b>
Project Tab.....	117
Options Tab.....	117
<b>Doc-To-Help</b> .....	<b>117</b>
Project Tab.....	118
Options Tab.....	118
<b>Document! X</b> .....	<b>118</b>
Project Tab.....	118
Options Tab.....	119
<b>Dr.Explain</b> .....	<b>119</b>
<b>Fast-Help</b> .....	<b>120</b>
<b>Flare</b> .....	<b>120</b>
<b>Help &amp; Manual</b> .....	<b>121</b>
Project Tab.....	121

Include Tab.....	121
Options Tab.....	122
<b>Helpinator .....</b>	<b>122</b>
Project Tab.....	123
Options Tab.....	123
<b>HelpMaker .....</b>	<b>123</b>
<b>HelpNDoc .....</b>	<b>123</b>
Project Tab.....	124
Output Tab.....	124
Output (More) Tab.....	124
Options Tab.....	125
<b>HelpScribble .....</b>	<b>125</b>
Project Tab.....	126
Options Tab.....	126
<b>HelpStudio .....</b>	<b>126</b>
Project Tab.....	126
Options Tab.....	127
<b>HyperText Studio .....</b>	<b>127</b>
<b>RoboHelp .....</b>	<b>128</b>
Project Tab.....	128
Options Tab.....	128
<b>6 Installers .....</b>	<b>129</b>
<b>Advanced Installer .....</b>	<b>129</b>
Project Tab.....	129
Properties Tab.....	129
Options Tab.....	129
<b>DeployMaster .....</b>	<b>130</b>
<b>ExpertInstall / QuickInstall / Tarma Installer / Tarma InstallMate .....</b>	<b>130</b>
Project Tab.....	131
Options Tab.....	131
<b>Inno Setup .....</b>	<b>131</b>
Project Tab.....	132
Options Tab.....	132
<b>InstallAnywhere .....</b>	<b>133</b>
Project Tab.....	133
Options Tab.....	133
<b>InstallAnywhere.NET .....</b>	<b>134</b>
Project Tab.....	134
Options Tab.....	134
<b>InstallAware .....</b>	<b>134</b>
Project Tab.....	135
Optimizations Tab.....	135
Options Tab.....	135
<b>Installer2Go / witem Installer .....</b>	<b>136</b>
<b>InstallShield .....</b>	<b>136</b>
Project Tab.....	137
Parameters Tab.....	138
Options Tab.....	138
Options (More) Tab.....	139
MSI Tab.....	140
<b>MSIStudio .....</b>	<b>140</b>
<b>NSIS .....</b>	<b>141</b>
Script Tab.....	141
Options Tab.....	141

<b>SetupBuilder</b> .....	<b>142</b>
Project Tab.....	142
Variables Tab.....	142
Options Tab.....	142
<b>Setup Factory / MSI Factory</b> .....	<b>143</b>
<b>Windows Installer</b> .....	<b>143</b>
Input Tab.....	144
Repair Tab.....	144
Options Tab.....	144
<b>Wise Setup</b> .....	<b>145</b>
Wise Tab.....	145
Options Tab.....	146
<b>WiX</b> .....	<b>146</b>
Project Tab.....	147
Compiler Tab.....	147
Linker Tab.....	147
Linker (more) Tab.....	148
Decompiler Tab.....	148
Harvester Tab.....	148
<b>7 Localization</b> .....	<b>149</b>
<b>Mutilizer</b> .....	<b>149</b>
Input Tab.....	149
Options Tab.....	149
<b>Sisulizer</b> .....	<b>150</b>
Input Tab.....	150
Options Tab.....	150
<b>8 Microsoft</b> .....	<b>151</b>
<b>Make VB6</b> .....	<b>151</b>
Project/Group Tab.....	152
Compatibility Tab.....	153
Versions Tab.....	153
Properties Tab.....	154
Options Tab.....	154
<b>Make VC6</b> .....	<b>155</b>
Project/Workspace Tab.....	155
Versions Tab.....	156
Properties Tab.....	157
Options Tab.....	157
<b>Make VS.NET</b> .....	<b>158</b>
Project/Solution Tab.....	159
Versions Tab.....	161
Properties Tab.....	162
Options Tab.....	162
<b>MSBuild</b> .....	<b>163</b>
Project Tab.....	164
Properties Tab.....	164
Output Tab.....	164
Output (More) Tab.....	165
Options Tab.....	165
<b>SourceSafe</b> .....	<b>166</b>
Database Tab.....	166
Flags Tab.....	167
Options Tab.....	168



<b>Team Build</b> .....	<b>168</b>
Build Tab.....	169
v2008+ Tab.....	169
Options Tab.....	169
<b>Team Foundation</b> .....	<b>170</b>
Command Tab.....	171
Options Tab.....	172
Changeset Tab.....	172
Checkin Tab.....	173
Difference Tab.....	173
Dir Tab .....	173
Get Tab.....	173
History Tab.....	174
Label Tab.....	174
Merge Tab.....	174
Resolve Tab.....	174
Shelve Tab.....	175
Status Tab.....	175
Workfold Tab.....	175
Workspace Tab.....	175
<b>Team Test</b> .....	<b>176</b>
Test Tab.....	176
Filter Tab.....	177
Team Explorer.....	177
Options Tab.....	178
<b>VS6 Get Version</b> .....	<b>178</b>
<b>VS.NET Get Version</b> .....	<b>178</b>
<b>Visual Studio Integration</b> .....	<b>179</b>
<b>Visual Studio &amp; .NET Macros</b> .....	<b>180</b>
<b>9 Microsoft .NET</b> .....	<b>180</b>
<b>App Packager</b> .....	<b>180</b>
Package Tab.....	180
Options Tab.....	181
<b>Export Type Library</b> .....	<b>181</b>
<b>GAC Install</b> .....	<b>181</b>
<b>Generate Resource Files</b> .....	<b>182</b>
Files Tab.....	182
Options Tab.....	183
<b>Import Type Library</b> .....	<b>183</b>
Import Tab.....	183
Options Tab.....	184
<b>Install .NET Services</b> .....	<b>184</b>
Services Tab.....	185
Options Tab.....	185
<b>Manifest Generator</b> .....	<b>185</b>
Manifest Tab.....	186
Settings Tab.....	186
Signing Tab.....	187
Options Tab.....	187
<b>NuGet</b> .....	<b>188</b>
Command Tab.....	188
Settings Tab.....	188
Options Tab.....	189
<b>PEVerify</b> .....	<b>189</b>

Verify Tab.....	189
Options Tab.....	189
<b>Strong Name Tool .....</b>	<b>190</b>
<b>WSDL .....</b>	<b>190</b>
WSDL Tab.....	191
Authentication Tab.....	191
Options Tab.....	192
<b>XSD .....</b>	<b>192</b>
XSD Tab.....	192
Options Tab.....	193
<b>10 Miscellaneous .....</b>	<b>193</b>
<b>Batch File .....</b>	<b>193</b>
Command Tab.....	194
Options Tab.....	194
<b>Kill Process .....</b>	<b>195</b>
<b>Play Sound .....</b>	<b>195</b>
<b>PowerShell .....</b>	<b>196</b>
Command Tab.....	196
Input Tab.....	197
Options Tab.....	197
<b>Read Registry .....</b>	<b>198</b>
Remote Tab .....	199
<b>Set Current Dir .....</b>	<b>199</b>
<b>Set Priority .....</b>	<b>199</b>
<b>Shut Down .....</b>	<b>200</b>
<b>Wait .....</b>	<b>200</b>
<b>Write Registry .....</b>	<b>201</b>
<b>11 Network .....</b>	<b>201</b>
<b>FTP .....</b>	<b>201</b>
Server Tab.....	202
Transfer Tab.....	203
Security Tab.....	204
Options Tab.....	205
<b>HTTP .....</b>	<b>206</b>
Server Tab.....	207
Command Tab.....	208
Options Tab.....	209
Security Tab.....	209
Form Tab.....	209
<b>Map Drive .....</b>	<b>210</b>
<b>Newsgroup Post .....</b>	<b>210</b>
Server Tab.....	210
Message Tab.....	211
Security Tab.....	211
<b>PLink Tunnel .....</b>	<b>212</b>
<b>Send Mail .....</b>	<b>212</b>
Server Tab.....	212
Message Tab.....	213
Security Tab.....	214
<b>Telnet .....</b>	<b>214</b>
Telnet Tab.....	215
Script Tab.....	216
Security Tab.....	216

<b>12 Server</b>	<b>217</b>
<b>ADO</b>	<b>217</b>
Connection Tab	217
Command Tab	218
<b>Amazon</b>	<b>218</b>
Command Tab	218
Options Tab	219
<b>Azure</b>	<b>219</b>
<b>COM Register</b>	<b>219</b>
<b>COM+ Application</b>	<b>220</b>
COM+ Properties Tab	221
COM+ Application Roles Tab	222
<b>COM+ Component</b>	<b>222</b>
COM+ Component Roles Tab	223
<b>IIS Virtual Dir</b>	<b>223</b>
IIS Tab	223
Options Tab	224
<b>IIS</b>	<b>224</b>
IIS Tab	224
Options Tab	225
Properties Tab	225
<b>Run Oracle Script</b>	<b>225</b>
Input Tab	225
Options Tab	226
<b>Run SQL</b>	<b>226</b>
Server Tab	226
Input Tab	227
Output Tab	227
Options Tab	228
<b>Service</b>	<b>228</b>
Service Tab	228
Properties Tab	229
<b>Web Deploy</b>	<b>230</b>
Source Tab	230
Destination Tab	230
Parameters Tab	231
Link Extensions Tab	231
Rules Tab	231
Skip Tab	231
Commands Tab	231
Miscellaneous Tab	232
Options Tab	232
<b>13 Test Driven Development</b>	<b>233</b>
<b>Ant</b>	<b>233</b>
Input Tab	233
Output Tab	233
Options Tab	234
<b>FxCop</b>	<b>234</b>
Input Tab	234
Output Tab	235
Options Tab	235
<b>Gallio</b>	<b>236</b>
Input Tab	236

Runner Tab.....	236
Reports Tab.....	237
Misc Tab.....	237
Options Tab.....	237
<b>Gradle .....</b>	<b>238</b>
Gradle Tab.....	238
Options Tab.....	238
<b>Maven .....</b>	<b>239</b>
Maven Tab.....	239
Projects Tab.....	239
Flags Tab.....	240
Options Tab.....	240
<b>NAnt .....</b>	<b>240</b>
Input Tab.....	241
Output Tab.....	241
Options Tab.....	241
<b>NCover .....</b>	<b>241</b>
Input Tab.....	242
Output Tab.....	242
Options Tab.....	242
<b>NDepend .....</b>	<b>243</b>
Project Tab.....	243
Options Tab.....	243
<b>NDoc .....</b>	<b>243</b>
Input Tab.....	244
Documentor Tab.....	244
Options Tab.....	244
<b>NUnit .....</b>	<b>245</b>
Input Tab.....	245
Include/Exclude.....	245
Output Tab.....	246
Options Tab.....	246
Options (More) Tab.....	247
<b>Sandcastle .....</b>	<b>247</b>
Project Tab.....	247
Options Tab.....	248
<b>14 Version Control .....</b>	<b>248</b>
<b>AccuRev .....</b>	<b>248</b>
Command Tab.....	248
Flags Tab.....	249
Options Tab.....	249
<b>Alienbrain .....</b>	<b>250</b>
Command Tab.....	250
Flags Tab.....	251
Responses Tab.....	252
Options Tab.....	252
<b>Bazaar .....</b>	<b>252</b>
Command Tab.....	252
Flags Tab.....	253
Options Tab.....	253
<b>ClearCase .....</b>	<b>253</b>
Command Tab.....	254
Options Tab.....	255
Checkout Tab.....	255

Deliver Tab.....	256
Import Tab.....	256
Make Activity Tab.....	257
Make Element Tab.....	257
Make Label Tab.....	257
Make LabelType Tab.....	258
Make View Tab.....	258
Make VOB Tab.....	258
Rebase Tab.....	259
Set Activity Tab.....	259
SetCS Tab.....	259
Update Tab.....	259
<b>CVS .....</b>	<b>260</b>
Repository Tab.....	260
Command Tab.....	261
Global Options Tab.....	261
<b>Dimensions .....</b>	<b>262</b>
Server Tab.....	263
Command Tab.....	263
Flags Tab.....	263
Attributes Tab.....	264
Options Tab.....	264
<b>Git .....</b>	<b>264</b>
Command Tab.....	265
Options Tab.....	265
<b>Mercurial .....</b>	<b>265</b>
Command Tab.....	266
Flags Tab.....	266
Options Tab.....	266
<b>Perforce .....</b>	<b>266</b>
Global Options Tab.....	267
Command Tab.....	268
Change/Form Fields Tab.....	269
Client Spec/Label Tab.....	269
<b>Plastic SCM .....</b>	<b>270</b>
Command Tab.....	270
Options Tab.....	270
<b>PureCM .....</b>	<b>270</b>
Server Tab.....	271
Command Tab.....	271
Flags Tab.....	271
Options Tab.....	271
<b>PVCS .....</b>	<b>272</b>
Command Tab.....	272
Flags Tab.....	273
Options Tab.....	273
<b>SCM Anywhere .....</b>	<b>273</b>
Database Tab.....	274
Command Tab.....	274
Flags Tab.....	274
Flags (More) Tab.....	275
Options Tab.....	275
<b>SourceAnywhere .....</b>	<b>276</b>
Database Tab.....	276

Command Tab.....	277
Flags Tab.....	277
Flags (More) Tab.....	277
Options Tab.....	278
<b>SourceOffSite .....</b>	<b>278</b>
Database Tab.....	279
Command Tab.....	279
Flags Tab.....	279
Options Tab.....	280
<b>Subversion .....</b>	<b>280</b>
Repository Tab.....	281
Subcommand Tab.....	282
Switches Tab.....	282
Options Tab.....	283
<b>Surround SCM .....</b>	<b>284</b>
Surround SCM Tab.....	284
Flags Tab.....	285
Options Tab.....	285
<b>Team Concert .....</b>	<b>285</b>
Server Tab.....	286
Command Tab.....	286
Options Tab.....	286
<b>Vault / Fortress .....</b>	<b>287</b>
Server Tab.....	287
Command Tab.....	288
Flags Tab.....	288
Options and Version 1 Tabs.....	289
<b>15 Virtual Machines .....</b>	<b>290</b>
<b>Hyper-V .....</b>	<b>290</b>
Hyper-V Tab.....	290
Options Tab.....	290
<b>Parallels .....</b>	<b>291</b>
<b>Virtual PC .....</b>	<b>291</b>
Virtual PC tab.....	291
Options tab.....	292
<b>Virtual Server .....</b>	<b>293</b>
Operation Tab.....	293
Parameters Tab.....	294
Text Tab.....	294
<b>VirtualBox .....</b>	<b>294</b>
<b>VMware Server .....</b>	<b>294</b>
<b>VMware Workstation / Server 2 / vSphere / Player .....</b>	<b>295</b>
VMware Tab.....	296
Server Tab.....	297
Options Tab.....	297
<b>Windows Virtual PC .....</b>	<b>297</b>
Virtual PC tab (COPY).....	298
Options tab (COPY).....	298
<b>16 Custom Action Details .....</b>	<b>299</b>
<b>Field Overrides .....</b>	<b>299</b>
<b>File Extensions .....</b>	<b>300</b>
<b>Default Properties .....</b>	<b>301</b>
<b>User-Defined Actions .....</b>	<b>301</b>

Action Registration .....	303
<b>Part IV Scripting</b> .....	<b>304</b>
1 Script Expressions .....	304
2 Script Events .....	305
3 System Scripts .....	306
Date/Time Functions .....	306
File Utility Functions .....	307
Miscellaneous Functions .....	308
Loop Support Functions .....	308
Obsolete Functions .....	309
<b>Part V Samples</b> .....	<b>316</b>
1 Advanced.bld Sample .....	317
2 Chain.bld Sample .....	317
3 Embarcadero.bld Sample .....	318
4 Files.bld Sample .....	318
5 Logging.bld Sample .....	318
6 Network.bld Sample .....	318
7 Prompt.bld Sample .....	318
8 Recurse.bld Sample .....	319
9 RegEdit.bld Sample .....	319
10 Script.bld Sample .....	319
11 SingleInstance.bld Sample .....	320
12 Server.bld Sample .....	320
13 XML.bld Sample .....	320
14 Version Control .....	320
ClearCase.bld Sample .....	320
ContinuousIntegration.bld Sample .....	320
CVS.bld Sample .....	321
Perforce.bld Sample .....	321
PVCS.bld Sample .....	321
StarTeam.bld Sample .....	321
Subversion.bld Sample .....	322
Surround SCM.bld Sample .....	322
Vault.bld Sample .....	322
15 Visual Studio .....	322
GetProjVer.bld Sample .....	322
SourceSafe.bld Sample .....	322
TDD.bld Sample .....	322
Team System.bld Sample .....	323
VStudio.bld Sample .....	323
16 Miscellaneous .....	323
SaveStatus.bld Sample .....	323
TestVFP.bld Sample .....	324
WebLauncher Sample .....	324

17	Advanced .....	324
	VisBuildPro Samples .....	324
	ObjectModel Samples .....	324
	VBPLogger Sample .....	325
<b>Part VI Commands and Procedures</b>		<b>325</b>
1	Automated Builds .....	325
	Console App .....	326
	GUI App .....	328
	Scheduling Builds .....	329
	Automatic Setup.....	330
	Manual Setup.....	330
	Continuous Integration .....	332
2	64-bit vs. 32-bit .....	333
3	Configuration Files .....	334
4	Project File Format .....	335
5	GUI Features .....	335
	Copy and Paste .....	335
	Create Desktop Shortcut .....	336
	Drag and Drop .....	336
	Explorer Interface .....	336
	Multiple Selection .....	337
	Navigation .....	337
	Tool Tip Features .....	338
	Undo/Redo .....	338
6	Event Logging .....	338
7	File Inclusion/Exclusion Matching .....	339
8	Predefined Macros .....	340
	System Macros .....	340
	Global Macros .....	342
9	Regular Expressions .....	343
10	Special Characters .....	344
11	Long Filenames .....	344
12	Usage Tips .....	344
<b>Part VII Object Model Reference</b>		<b>346</b>
1	Application Object .....	347
	Context Property .....	348
	Options Property .....	348
	Project Property .....	348
	Macros Property .....	349
	Scripts Property .....	349
	Steps Property .....	350
	ExpandMacros Function .....	350
	ExpandMacrosAndScript Function .....	351
	FindMacro Function .....	351
	GlobalSubroutineStepsLoaded Event .....	351
	Initialize Method .....	352



Uninitialize Method .....	352
ProjectModified Event .....	353
ProjectSaved Event .....	353
PromptMacroValue Event .....	353
MacroModified Event .....	354
<b>2 Builder Object .....</b>	<b>354</b>
App Property .....	356
BuildIndex Property .....	356
BuildThread Property .....	356
CallStack2 Property .....	357
CancelEvent Property .....	357
CompletionStatus Property .....	358
Debugging Property .....	358
FailedStep Property .....	358
LastStep Property .....	359
LaunchType Property .....	359
MaxStopWait Property .....	360
ProcessID Property .....	360
StartTime Property .....	360
Status Property .....	361
StepType Property .....	361
EvaluateRule method .....	361
ExpandMacros Method .....	362
GetFileContents Method .....	363
Initialize Method .....	363
LogFileContents Method .....	363
LogMessage Method .....	364
LogMessage2 Method .....	364
LogMessageEx Method .....	365
Pause Method .....	365
RegisterKey Method .....	366
ResetBuildStatus Method .....	366
Resume Method .....	366
RunProgram Method .....	367
RunProgram2 Method .....	368
RunProgramEx Method .....	369
RunProgramEx2 Method .....	371
Start Method .....	371
StartEx Method .....	372
StartEx2 Method .....	372
Stop Method .....	374
SyncBuild Method .....	374
SyncBuildEx Method .....	375
Uninitialize Method .....	376
BuildStarting Event .....	376
BuildDone Event .....	377
StepStarting Event .....	377
StepStarted Event .....	377
StepDone Event .....	378
BuildMessage Event .....	378
<b>3 Macro Object .....</b>	<b>379</b>
Category Property .....	379
Name Property .....	379

Value Property .....	379
Parameters Property .....	380
AddAsEnvVar Property .....	380
<b>4 Macros Collection .....</b>	<b>381</b>
Type Property .....	381
Item Property .....	381
Name Property .....	382
Add Method .....	382
AddEx Method .....	383
Remove Method .....	384
Load Method .....	384
Save Method .....	384
<b>5 Options Object .....</b>	<b>385</b>
AlwaysShowGUIApps Property .....	385
BuildFailureStepsOnCancel Property .....	386
CallStepStartingScriptEventOnSkippedStep Property .....	386
CacheActions Property .....	386
CaseSensitiveBuildRuleComparisons Property .....	386
ConfigFilesPath Property .....	386
ConvertOutputDoubleQuotes Property .....	387
DefaultScriptEngine Property .....	387
DelLogFileOnBuild Property .....	387
DelTempMacrosAfterBuild Property .....	387
EchoChainedConsoleOutput Property .....	388
EnableEventErrorLogging Property .....	388
EncryptPasswordProperties .....	388
EnvVarsInSystemMacros Property .....	388
EscapeSpecialCharactersInOutput Property .....	388
FailBuildWhenDoneIfStepsFailed Property .....	389
HidePasswordPropertyValues Property .....	389
LogAllFailureStepOutput Property .....	389
LogDefaultStepProperty Property .....	389
LogBuildRuleEval Property .....	390
LogFilename Property .....	390
LogFormat Property .....	390
LogLevel Property .....	390
MacroExpandBuildTimeout Property .....	391
MacroExpandTooltipTimeout Property .....	391
MaxDefaultPropertyLogLength Property .....	391
MaxMacroLengthExpand Property .....	391
MaxStepOutputLength Property .....	392
NestBuildRules Property .....	392
NestIncludeInBuild Property .....	392
PersistBuildStatus Property .....	392
PreventDuplicateEventLog Property .....	393
PsExecCmd Property .....	393
ReevaluateAllBuildRules Property .....	393
RunProgramInputWaitTimeout Property .....	393
RunProgramOutputBufferSize Property .....	393
RunProgramOutputWaitTimeout Property .....	394
RunProgramRedirInputChunkSize Property .....	394
ScriptTypeLibraries Property .....	394
SetProjectDirectory Property .....	394

StripLogLinefeedChar Property .....	395
TextLogSkippedSteps Property .....	395
TextLogStepEvents Property .....	395
TextLoggerCacheMessages Property .....	395
TextLoggerCacheSeconds Property .....	396
UniqueEncryptionIV Property .....	396
UseLogonCreateProcessAsUser Property .....	396
UseLogonOptionWithSyncBuildEx Property .....	396
UseUTF8ForConsoleApps Property .....	397
UseUSEngishLocaleForScriptEngine .....	397
WriteBOM Property .....	397
XMLLoggerCacheSeconds Property .....	397
XMLLoggerCacheMessages Property .....	398
Load Method .....	398
Save Method .....	398
<b>6 Project Object .....</b>	<b>399</b>
Macros Property .....	399
Name Property .....	399
Parent Property .....	399
Password Property .....	400
Scripts Property .....	400
Steps Property .....	400
Comments Property .....	401
IsModified Property .....	401
FindStep Method .....	401
Load Method .....	402
Save Method .....	402
New Method .....	403
<b>7 Script Object .....</b>	<b>404</b>
Language Property .....	404
Value Property .....	404
<b>8 Scripts Collection .....</b>	<b>404</b>
Type Property .....	405
Item Property .....	405
Add Method .....	406
Remove Method .....	406
<b>9 Step Object .....</b>	<b>407</b>
Action Property .....	407
BuildStatus Property .....	407
Checked Property .....	408
ContinueOnFailure Property .....	408
Description Property .....	409
Expanded Property .....	409
ExpProperty Property .....	409
FailureSubroutine Property .....	410
ID Property .....	410
Indent Property .....	410
Index Property .....	411
BuildFailureStepsOnFailure Property .....	411
Name Property .....	411
NoLogging Property .....	412
Property Property .....	412
Property2 Property .....	413

PropertyEx Property .....	413
Properties Property .....	414
RuleComparison Property .....	414
RuleExpression Property .....	415
RuleCompareTo Property .....	415
RuleRepeat Property .....	415
Script Property .....	416
<b>10 Steps Collection .....</b>	<b>416</b>
Type Property .....	416
Item Property .....	417
Count Property .....	417
_NewEnum Property .....	418
Add Method .....	418
Clear Method .....	418
Find Method .....	419
Remove Method .....	419
<b>11 WScript Object .....</b>	<b>419</b>
CreateObject Method .....	420
Echo Method .....	420
Sleep Method .....	420
<b>12 Threading .....</b>	<b>421</b>
 <b>Part VIII Purchase &amp; Support .....</b>	 <b>421</b>
<b>1 Order Info .....</b>	<b>421</b>
<b>2 How To Register .....</b>	<b>421</b>
<b>3 Technical Support .....</b>	<b>422</b>
<b>4 Revision History .....</b>	<b>422</b>
 <b>Index .....</b>	 <b>423</b>

# 1 Introduction

## 1.1 Overview

[Visual Build](#) enables developers and build masters to create an automated, repeatable process for building and deploying software. Visual Build can automate common build tasks such as:

- Retrieving source code, checking in changes, and labeling in version control systems
- Building Microsoft, Embarcadero and other source code projects
- Setting and retrieving project versions
- Compiling help projects
- Creating installers
- Copying, deleting, renaming, listing, and setting file attributes
- Burning files to CD or DVD
- Creating, updating and extracting from ZIP files
- Deploying files to or from a web site, FTP server, or network path
- Securely transferring information using SFTP, SSH, or SSL
- Deploying applications that utilize Microsoft COM+, SQL Server, and IIS
- Manipulating virtual machines
- Sending e-mail and newsgroup posts to notify of build success or failure
- Logging of all build activities and generating log reports
- Performing automated, scheduled, remote, continuous, and online builds
- Performing test-driven development
- Writing to and reading from text, XML, and INI files
- Search and replace and read text from files
- Reading and writing registry settings
- Performing build operations in parallel
- Registering type libraries, components, and .NET assemblies
- Running unit test cases and generating documentation
- Starting and stopping Windows services
- Much, much more

It executes all those tedious steps that must be performed over and over again, while ensuring that all steps have been successfully completed. A detailed status of each step is displayed during the build and optionally logged to a file. If any step fails, Visual Build pinpoints the error and then continues from the point of failure after the problem has been corrected.

Visual Build provides built-in support for **Microsoft** Visual Studio and **Embarcadero / Borland / CodeGear** Delphi, RAD Studio, C++Builder, and more.

It's easy to put Visual Build to work doing all that manual labor for you so you can focus on more important things. It will help you automate the build process without a huge commitment of time, and it works with the tools you're using today.

[Click here to take a tour of the application.](#)

## 1.2 Why Visual Build?

### Save Time and Money

Automating your builds with Visual Build can save many hours of manual effort every month and easily pay for itself in the first month alone.

## Improve Product Quality

By implementing a regular build process, your team's software quality will improve and bugs will be found and repaired more quickly.

## Avoid Boring Jobs

Visual Build performs the menial, repetitive tasks so you can focus on the fun, challenging stuff.

## Eliminate Key-Person Dependencies

"Tribal knowledge" about the build procedure (often kept only in people's heads) will be replaced by a documented build framework.

## Create a Permanent Build Record

XML and HTML project files and log files, which become part of your project's source code (and can be maintained in source control), provide a persistent, documented record for your builds.

## Have Fun!

Visual Build is the fun, easy way to automate your software builds. Give it a try today.

## 1.3 New Features

### 1.3.1 Version 4

- Tree display of steps, with ability to expand and collapse steps
- Custom action screens and components for many third party development tools to simplify the creation of build scripts
- Drag-and-drop adding of steps for custom actions
- Extensive debugging features (single-stepped execution, breakpoints, pause, etc.)
- Colorization of build output for easier navigation
- Editing and execution of Windows Script is fully integrated into Visual Build
- Full support for Microsoft Visual Studio .NET (of course, Visual Studio 6.0 is still supported as well)
- Ability to generate XML log files
- Many new user-configurable options
- Shortcuts to easily jump to matching output for a step and vice versa, Go to a top-level step, open a filename displayed in the build output, and more (see View menu).
- Multiple failure steps can be defined in a project
- Subroutine steps allow modularizing code within a single project

- The build status of the steps can be saved in the project file
- COM Automation interface to access all application and build components programmatically
- Console application to run locally or remotely without requiring interactive desktop access or a GUI
- Global macros and application options are stored in XML files instead of a user-specific registry key
- Single-click testing of steps
- Option for macros to be added to environment variables
- Search and replace in macros and script as well as steps
- Enhancements to tool tip feature
- Ability to specify multiple ranges of successful exit codes for the Run Program custom action

### 1.3.2 Version 5

- Ability to generate HTML log files
- Integration with Windows Scheduled Tasks applet for easy scheduling of builds
- Enhancements to scripting capabilities Windows Script to add user-defined code to build events
- End-user creation of custom action screens and components
- Additional keyboard shortcuts and user options
- New *True* comparison for conditional build rules (useful for script expressions in rules)
- Child projects started from a VisBuildPro Project action are launched minimized if the parent project is minimized
- Support for global subroutines that can be called from multiple projects
- Enhancements to project properties dialog
- Enhancements to drag/drop and copy/paste features
- Editing of multiple step properties at once via multiple selection
- Write to Windows Event Viewer when scheduled builds fail
- Customization of the main and action toolbars now supported
- Support for entry and display of Unicode text
- Normal operation does not require power user privileges (although some build operations may require power or even admin user privileges)
- Custom action screens and components for more third party development tools and several enhancements to existing actions:
  - \* COM+ Component

- \* Copy Files
- \* Help & Manual
- \* InstallShield
- \* InstallAnywhere (new)
- \* Make Delphi (new)
- \* Make JBuilder (new)
- \* Make VS.NET
- \* Perforce (new)
- \* Replace in File
- \* Run Program
- \* Run SQL (new)
- \* Send Mail
- \* Sign Code (new)
- \* SourceSafe
- \* Set Current Dir (new)
- \* Set Macro
- \* Subversion (new)
- \* Telnet
- \* Transform XML Log (new)
- \* Surround SCM (new)
- \* UNZIP Files (new)
- \* Vault (new)
- \* Wise
- \* Write File
- \* Write Registry
- \* ZIP Files (new)

*Note:* See the [history page](#) on the web site for full details.

### 1.3.3 Version 6

- Full undo/redo capability
- Dockable panes, toolbars, menus, and themes
- Full support for Windows Vista
- Full keyboard shortcut customization
- Forward/back navigation and support for dedicated mouse/keyboard forward/back buttons/keys
- Quick navigation to the last build or script error
- System tray functionality
- The ability to disable logging of step output for individual steps
- Ability to run a specified failure subroutine on step failure
- Script editor enhancements (code completion, toolbar and menu bar, incremental search, block comment, etc.)
- WebLauncher sample
- Object model enhancements
- Project-level step script events



- Improvements to Insert Macro dialog
- GUI designer for user actions
- Enhancements to user action GUI capabilities and methods of registration
- Support persisting of step expand/collapse state in project file
- New *False* comparison for conditional build rules
- Exit subroutine or build step failure option
- Custom action screens and components for more products and enhancements to existing actions:
  - \* AccuRev (new)
  - \* Alienbrain (new)
  - \* ADO (new)
  - \* Advanced Installer (new)
  - \* Ant (new)
  - \* Batch File (new)
  - \* Burn CD/DVD (new)
  - \* ClearCase (new)
  - \* C++Builder (new)
  - \* COM Register (support registering .NET assemblies for COM Interop)
  - \* COM+ Application
  - \* COM+ Component
  - \* Copy Files (Skip files that already exist in destination or are older than destination, limit by size, date, advanced wildcards)
  - \* Create Folder (new)
  - \* Delete Files (new)
  - \* Delete Folder (new)
  - \* Delphi
  - \* DeployMaster (new)
  - \* Doc-O-Matic (new)
  - \* Doc-To-Help (new)
  - \* Exit (new)
  - \* ExpertInstall (new)
  - \* Export Type Library (new)
  - \* Fast-Help (new)
  - \* Flare (new)
  - \* FTP (recursive copies, incremental copies, synchronization, secure transfers, additional wildcard options)
  - \* FxCop (new)
  - \* Generate Resource Files (new)
  - \* Help & Manual
  - \* HelpScribble (new)
  - \* HelpStudio (new)
  - \* HTTP (new)
  - \* HyperText Studio (new)
  - \* IIS Virtual Dir (new)
  - \* Import Type Library (new)
  - \* Install .NET Services (new)
  - \* InstallAnywhere.NET (new)
  - \* InstallShield
  - \* InstallAWARE (new)
  - \* List Files (new)
  - \* Map Drive (new)

- \* Kill Process (new)
- \* Make VS 2005 (new)
- \* Make VS.NET
- \* MSBuild (new)
- \* MSIStudio (new)
- \* Multilizer (new)
- \* NAnt (new)
- \* NCover (new)
- \* NDoc (new)
- \* Newsgroup Post (new)
- \* NSIS (new)
- \* NUnit (new)
- \* Oracle (new)
- \* Perforce
- \* PEXVerify (new)
- \* Play Sound (new)
- \* Plink Tunnel (new)
- \* PowerShell (new)
- \* Process Files (additional matching, logging and debugging capabilities)
- \* PVCS (new)
- \* Rename Files (new)
- \* Replace in File (encoding/BOM enhancements, regex aids)
- \* RoboHelp (new)
- \* Run Program
- \* Sandcastle (new)
- \* Send Mail (secure connections, return receipts)
- \* Set File Attributes (new)
- \* Set Priority (new)
- \* SetupBuilder (new)
- \* Setup Factory (new)
- \* Shut Down (new)
- \* StarTeam (new)
- \* Strong Name Tool (new)
- \* Surround SCM
- \* Telnet (secure connections)
- \* UNZIP Files (strong encryption support)
- \* Virtual PC (new)
- \* Virtual Server (new)
- \* VMWare Server (new)
- \* VMWare Workstation (new)
- \* Visual Studio Team System -- Team Build, Team Foundation, and Team Test (new)
- \* Wait (wait on processes)
- \* Windows Installer (new)
- \* WiX (new)
- \* Write File (encoding/BOM enhancements)
- \* Write XML (new)
- \* WSDL (new)
- \* XSD (new)
- \* ZIP Files (strong encryption support, additional wildcard options)

- More...

Note: See the [history page](#) on the web site for full details.

### 1.3.4 Version 7

- Build call stack

- Watch window
- Windows 7 and Windows 2008 R2 compatibility
- Linux and Intel Mac OS X compatibility
- Enhanced support for 64-bit Windows
- Preview and print step, macros, and output panes as displayed in the GUI
- Easier navigation in larger projects (Enhancements to build output navigation and Go To Step dialog).
- Hide actions from Actions pane
- Additional logging options, build output filtering, and build output coloring
- Encryption of protected step properties and macros
- Retry on step failure
- Pause before and/or after building a step
- Advanced options for all Run Program-derived actions
- Remote execution of all Run Program-derived actions
- Additional remote execution options
- Custom action screens and components for more products and enhancements to existing actions:
  - \* 7-Zip (new)
  - \* AccuRev
  - \* ADO
  - \* Advanced Installer
  - \* Batch File
  - \* Bazaar (new)
  - \* Burn CD/DVD (support importing previous sessions, and filtering on size, date, and attributes)
  - \* Copy Files
  - \* COM+ Application
  - \* COM+ Component
  - \* Delete Files (filter on size, date, and attributes)
  - \* DeployMaster
  - \* Dimensions (new)
  - \* Doc-O-Matic
  - \* Doc-To-Help
  - \* Document! X (new)
  - \* Dr.Explain (new)
  - \* ExpertInstall
  - \* Fast-Help
  - \* Flare
  - \* Fortress
  - \* FTP (SFTP support, delete files option)
  - \* Fortress (new)
  - \* Gallio (new)
  - \* Git (new)
  - \* Help and Manual

- \* Helpinator (new)
- \* HelpMaker (new)
- \* HTTP (option to log % complete progress; put and get in same step; specify form, request header, and cookie values)
- \* IIS (new)
- \* Inno Setup
- \* InstallAware
- \* InstallAnywhere
- \* Installer2Go (new)
- \* InstallShield (support v2009 and 2010)
- \* List Files (option to list empty folders)
- \* Loop (new)
- \* Make Delphi / C++Builder / RAD Studio (support v2009 and 2010)
- \* Make Delphi Prism (new)
- \* Make VB6
- \* Make VS.NET / 2005 / 2008 / 2010 (support VS 2008 and 2010, parallel builds with MSBuild, and additional project types)
- \* Maven (new)
- \* Mercurial (new)
- \* MSI Factory (new)
- \* Multilizer
- \* NAnt
- \* NCover
- \* NDepend
- \* NUnit
- \* Perforce
- \* Plastic SCM (new)
- \* PowerShell
- \* Process Files (filter on size, date, and attributes)
- \* PureCM (new)
- \* Read File (new)
- \* Read INI (new)
- \* Read Registry (new)
- \* Read XML (new)
- \* Rename Files (filter on size, date, and attributes, support renaming of folders)
- \* RoboHelp
- \* Run Oracle Script
- \* Run Program
- \* Run SQL
- \* Sandcastle
- \* SCM Anywhere (new)
- \* Service (create, configure, pause, resume, query, and delete operations)
- \* Set File Attributes (filter on size, date, and attributes)
- \* Set Macro
- \* SetupBuilder
- \* Setup Factory (add MSI Factory support)
- \* Sign Code (Options to retain original modification timestamp and not re-sign if already signed)
- \* Sisulizer
- \* SourceAnywhere (new)
- \* SourceOffSite (new)
- \* StarTeam
- \* Strong Name Tool
- \* Subversion
- \* Surround SCM (support 2008 thru 2010)
- \* Tarma InstallMate (new)
- \* Team Build (support VS 2008 and 2010)
- \* Team Foundation (support VS 2008 and 2010)
- \* Team Test (support VS 2008 and 2010)

- \* Transform XML Log
- \* Telnet (SSH support)
- \* UNZIP Files (support .gz and .tar files)
- \* Vault
- \* VisBuildPro Project
- \* VMware Server 2 / vSphere / Player (new)
- \* VMware Server
- \* Wait
- \* Web Deploy (new)
- \* Windows Virtual PC (new)
- \* Wise
- \* Write Registry
- \* Write XML
- \* ZIP Files (support .gz and .tar files)

- More...

*Note:* See the [history page](#) on the web site for full details.

### 1.3.5 Version 8

- 64-bit (x64) editions of GUI app, console app, and object model
- Windows 8 and Server 2012 compatibility
- Support for Visual Studio 2012 and Visual Studio 2013
- Show build progress, pause, and failure in Windows taskbar (Windows 7 and later)
- Option to save project files in compressed, encrypted format (.bldx extension)
- Option to fail overall build if any steps failed (but continued) during build
- Ability to specify an edit password to allow a project to be built but not modified without the password
- Specify the log level separately for the console app, GUI app, and XML and text log files
- Honor text log file options for console app output
- Options to configure behavior of the Go to step dialog for easier navigation in large projects
- Option to play a sound when stopping at a breakpoint
- Separate build status for terminated actions
- Support nesting of Loop and Process Files actions
- Honor retry option when step build status set to failure in vbld\_StepDone script event
- Make Call Stack accessible to scripting
- Allow renaming and deleting of categories in Macros pane
- Improved performance when building many iterating steps in GUI app
- Reduced file size when saving project files

- Version-independent ProglDs for the object model (Application and Builder)
- Custom action screens and components for more products and enhancements to existing actions:
  - \* AccuRev
  - \* Advanced Installer
  - \* App Packager
  - \* Create Folder
  - \* DeployMaster
  - \* Doc-To-Help
  - \* Document! X
  - \* Enhanced Unzip Files (new)
  - \* Enhanced Zip Files (new)
  - \* Exit
  - \* File Checksum
  - \* Flare
  - \* Gradle (new)
  - \* Help & Manual
  - \* Helpinator
  - \* HelpNDoc (new)
  - \* HelpStudio
  - \* HTTP
  - \* Hyper-V (new)
  - \* IIS Virtual Dir
  - \* InstallAnywhere
  - \* InstallAware
  - \* InstallMate
  - \* InstallShield
  - \* Make Delphi / C++Builder / RAD Studio
  - \* Make VS 2010
  - \* Make VS 2012 (new)
  - \* Make VS 2013 (new)
  - \* Map Drive
  - \* Maven
  - \* Mercurial
  - \* MSBuild
  - \* Multilizer
  - \* NAnt
  - \* NDepend
  - \* NSIS
  - \* NuGet (new)
  - \* NUnit
  - \* Parallels (new)
  - \* Perforce
  - \* Plastic SCM
  - \* PowerShell
  - \* PureCM
  - \* Read XML
  - \* RoboHelp
  - \* Run Program (run remote command with elevation)
  - \* Run SQL
  - \* Service
  - \* Setup Factory
  - \* SetupBuilder
  - \* Sign Code
  - \* Sisulizer
  - \* SourceAnywhere

- \* StarTeam
- \* Subversion
- \* Surround SCM
- \* Team Concert
- \* Team Foundation
- \* Telnet
- \* Transform XML Log
- \* Vault
- \* VirtualBox (new)
- \* VMware Workstation
- \* Web Deploy
- \* Windows Installer
- \* WiX
- \* Write INI
- \* Write XML

- More...

*Note:* See the [history page](#) on the web site for full details.

### 1.3.6 Version 9

- Windows 10 compatibility
- Modeless properties pane for editing step and macro properties
- Support for Visual Studio 2015 and Team System 2015
- Build profiles to choose which sets of steps to build
- Command to open a Command Prompt or console to the selected folder
- Option to remember previous step selections when opening projects
- Support drag/drop from Windows Explorer onto Step Properties dialog controls
- Support code completion in Script Editor in 64-bit edition
- Add step number column to Step Panes
- Option to format Watches pane values
- Options to cache writes to text and XML log (enabled by default)
- Script event when loading a project
- Include macro type in Action column of Step panes for Set Macro action
- Save encrypted properties and macros using AES-256 algorithm
- Object model: 1) Make Step.Action property writeable; 2) new option related to SyncBuildEx; 3) Step.Property2 property for setting encrypted values; 4) *Options.PsExecCmd* property.
- Hide actions for products that are not installed by default (to show all hidden actions, right-click in the Actions pane and choose *Show Hidden*)

- Additional options when specifying credentials for processes
- Options to configure timeout when expanding macros and script
- Custom action screens and components for more products and enhancements to existing actions:
  - \* File processing actions: 1) performance improvements and 2) automatic support for extended-length paths
  - \* 7-Zip
  - \* Advanced Installer
  - \* Amazon (new)
  - \* Azure (new)
  - \* Copy Files
  - \* Delete Files
  - \* Document! X
  - \* Dr.Explain
  - \* Enhanced Zip Files
  - \* Enhanced Unzip Files
  - \* Fast-Help
  - \* Flare
  - \* FTP
  - \* Git
  - \* Gradle
  - \* Help & Manual
  - \* HelpStudio
  - \* Inno Setup
  - \* InstallAnywhere
  - \* InstallAware
  - \* InstallShield
  - \* Loop
  - \* Manifest Generator (new)
  - \* Make Delphi / C++Builder / RAD Studio
  - \* Make VS 2015 (new)
  - \* Multilizer
  - \* NDepend
  - \* NuGet
  - \* NUnit
  - \* Perforce
  - \* PowerShell
  - \* Process Files
  - \* Read File
  - \* Read INI
  - \* Read Registry
  - \* Read XML
  - \* RoboHelp
  - \* Run Program
  - \* Run SQL
  - \* Set File Attributes
  - \* SetupBuilder
  - \* Sign Code
  - \* SourceAnywhere
  - \* StarTeam
  - \* Surround SCM
  - \* Transform XML Log
  - \* Vault
  - \* Virtual Box
  - \* VMware Player
  - \* VMware Workstation



- More...

*Note:* See the [history page](#) on the web site for full details.

## 1.4 License Agreement

Visual Build™ Professional  
Copyright © 1999-2015 Kinook Software, Inc.  
All Rights Reserved.

### License Agreement

This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Kinook™ Software for the software product identified above, which includes computer software and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT"). By installing, copying, or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree to the terms of this EULA, do not install or use the SOFTWARE PRODUCT.

#### SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

**1. GRANT OF LICENSE.** This EULA grants you the following rights:

**Software.** Each license of the SOFTWARE PRODUCT may either be used by a single person who uses the SOFTWARE PRODUCT personally on one or more computers, or installed on a single computer used non-simultaneously by multiple people, but not both. This is not a concurrent use license.

If a Site License has been purchased, you may install the SOFTWARE PRODUCT on an unlimited number of computers at a single site. A site is defined as all company locations within a 20-mile radius. Site licenses do not cover subsidiary companies or customers of a company.

**Storage/Network Use.** You may also store or install a copy of the computer software portion of the SOFTWARE PRODUCT to allow your other computers to use the SOFTWARE PRODUCT over an internal network, and distribute the SOFTWARE PRODUCT to your other computers over an internal network. However, you must acquire a license for the SOFTWARE PRODUCT for each computer on which the SOFTWARE PRODUCT is used or to which it is distributed. A license for the SOFTWARE PRODUCT may not be shared or used concurrently on different computers.

**Evaluation Use.** You may use the evaluation version of the SOFTWARE PRODUCT for evaluation purposes without charge for a period of 30 days. If you use the SOFTWARE PRODUCT after the 30-day evaluation period, a license for the non-evaluation version must be purchased. When payment is received you will be sent a license key to enable the full functionality of the SOFTWARE PRODUCT.

#### 2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.

Limitations on Reverse Engineering, Decompilation, and Disassembly. You may not reverse engineer, decompile, or disassemble the SOFTWARE PRODUCT, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

**Separation of Components.** The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

**Rental.** You may not rent or lease the SOFTWARE PRODUCT.

**Software Transfer.** You may not transfer your rights under this EULA.

**Termination.** Without prejudice to any other rights, Kinook™ Software may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

### **3. COPYRIGHT.**

All title and copyrights in and to the SOFTWARE PRODUCT, the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by Kinook™ Software. The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material except that you may install the SOFTWARE PRODUCT on a single computer provided you keep the original solely for backup or archival purposes.

### **4. EXPORT CONTROLS.**

You agree to comply with all applicable U.S. export control laws and regulations, including without limitation, the laws and regulations administered by the United States Department of Commerce and the United States Department of State.

## **Disclaimer of Warranty**

### **NO WARRANTIES.**

Kinook™ Software expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT and any related documentation is provided "as is" without warranty of any kind, either express or implied, including, without limitation, the implied warranties or merchantability, fitness for a particular purpose, or non infringement. The entire risk arising out of use or performance of the SOFTWARE PRODUCT remains with you.

### **NO LIABILITY FOR DAMAGES.**

In no event shall Kinook™ Software be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this product, even if Kinook™ Software has been advised of the possibility of such damages.

## **1.5 Installation**

### **System Requirements**

- Visual Build runs on Microsoft [Windows](#) Vista, 7, 8, 8.1, 10, and Server 2003 thru Server 2012 R2 (32- and 64-bit editions).
- Installation must be performed by an account with admin privileges.
- About 35 MB free disk space.
- 256-color and 800 x 600 resolution display settings (Recommended: 16-bit color and 1024 x 768 resolution or greater).

*Note:* Visual Build is available in 32-bit and 64-bit editions (via separate downloads). The **32-bit** edition is compatible with 32-bit editions of Windows as well as 64-bit editions of Windows with the WoW64 subsystem (which is installed with most flavors of Windows but is optional on Windows Server 2008 R2 Server Core and later). The **64-bit** edition is a native x64 executable that does not use the WoW64 subsystem and must be installed on a 64-bit edition of Windows.

### **Installation Procedure**

1. Run the setup application (`VisBuildProEval.exe` or `VisBuildProX64Eval.exe` for the 64-bit edition).

2. Follow the instructions on the installation dialogs. Please note that the software can only be installed for the number of computers or users it has been licensed for. Multiple and site licenses are available. Visit the [Kinook Software web site](#) for more information.
3. Once Visual Build has been successfully installed, you will find new start menu icons for Visual Build Professional. Entries for the application, help, and samples will be available.

## Build-Only Installation

If the *Minimal installation* option is chosen, only the console application (not the GUI app or any start menu shortcuts) and required build components will be installed. This can be useful for providing a read-only/build-only installation of Visual Build, since no GUI for editing project files will be available. However, a license is still required and the license must be registered (see below).

## Unattended Installation and Registration

The Visual Build installer accepts the following optional command line parameters:

/SILENT, /VERYSILENT	The wizard and background window are not displayed but the installation progress window is. When a setup is very silent, the progress window is not displayed. Error messages during installation are displayed. If a restart is necessary and the /NORESTART command isn't used and Setup is silent, it will display a <i>Reboot now?</i> message box. If it's very silent, it will reboot without asking.
/SUPPRESSMSG BOXES	Suppresses messageboxes. Only has an effect when combined with /SILENT and /VERYSILENT. The default response in situations where there's a choice is: -Yes in a 'Keep newer file?' situation. -No in a 'File exists, confirm overwrite.' situation. -Abort in Abort/Retry situations. -Cancel in Retry/Cancel situations. -Yes (=continue) in a DiskSpaceWarning/DirExists/DirDoesntExist/NoUninstallWarning/ExitSetupMessage/ConfirmUninstall situation. -Yes (=restart) in a FinishedRestartMessage/UninstalledAndNeedsRestart situation.
/LOG	Creates a log file in the user's TEMP directory detailing file installation and [Run] actions taken during the installation process. This can be a helpful debugging aid. For example, if you suspect a file isn't being replaced when you believe it should be (or vice versa), the log file will tell you if the file was really skipped, and why. The log file is created with a unique name based on the current date. (It will not overwrite or append to existing files.) The information contained in the log file is technical in nature and therefore not intended to be understandable by end users. Nor is it designed to be machine-parseable; the format of the file is subject to change without notice.
/LOG="filename"	Same as /LOG, except it allows specifying a fixed path/filename to use for the log file. If a file with the specified name already exists it will be overwritten. If the file cannot be created, Setup will abort with an error message.
/NOCANCEL	Prevents cancelling the installation process by disabling the Cancel button and ignoring clicks on the close button. Useful along with /SILENT or /VERYSILENT.
/NORESTART	Does not reboot even if necessary.
/DIR="x:\dirname"	Overrides the default directory name displayed on the <i>Select Destination Location</i> wizard page. A fully qualified pathname must be specified.
/GROUP="folder name"	Overrides the default folder name displayed on the <i>Select Start Menu Folder</i> wizard page.

/NOICONS	Initially checks the <i>Don't create any icons</i> checkbox on the Select Start Menu Folder wizard page.
/TYPE=type	Specifies a setup type for the <i>Select Components</i> wizard page ( <i>full, compact, or minimal</i> ).
/COMPONENTS="comma separated list of component names"	Specifies components for the <i>Select Components</i> wizard page ( <i>required, application, samples, addins, and/or tools</i> ), selecting a <i>custom</i> installation type. Only the specified components will be selected; the rest will be unchecked. If a component name is prefixed with a "!" character, the component will be unchecked.
/TASKS="comma separated list of task names"	Specifies a list of tasks from the <i>Select Additional Tasks</i> wizard page ( <i>desktopicon, quicklaunchicon, assoc, and/or explorer</i> ) that should be selected. Only the specified tasks will be checked; the rest will be unchecked. If a task name is prefixed with a "!" character, the task will be unchecked.

Automated entry of license info requires using the object model, either within Visual Build via the `vblld_BuildStarting` script event (VBScript) like:

```
Builder.RegisterKey "name + license count", "license key"
```

or outside Visual Build (i.e., in a .vbs file):

```
Set bld = CreateObject("VisBuildSvr.Builder")
bld.RegisterKey "name + license count", "license key"
```

Alternatively, run Visual Build as *administrator* and register, or create a text file named `C:\Program Files\VisBuildPro9\System\VisBuildSvr.ini` (in the Visual Build installation *System* subfolder) with contents like:

```
[License]
PortableKey=license name + count,license key
```

*Note:* Replace "name + license count" and "license key" above with the actual license name + count and key from your license info email.

## 1.6 Upgrading from Older Versions

### 1.6.1 Visual Build 2/3

Existing projects created in Visual Build 2 and 3 can be loaded by Visual Build 4+. However, once saved in Visual Build, the projects will not load correctly into earlier versions. There are also a few compatibility differences to be aware of when upgrading.

#### Changes that may require project updates

- Environment variables are no longer reloaded by Visual Build when they are changed by an external program. Existing projects that use a utility such as SETENV to modify environment variables should be modified to instead use a macro or the Set Macro action, marking the macro to be added to the environment variables.
- Existing application options will not be copied from a previous version and must be set in the Visual Build options dialogs. Application options cannot be configured by modifying the registry (they are now stored in an XML file). To modify options within a project, use the automation object model Application object within a Run Script action. Some defaults settings are different than the Visual Build 3 defaults.
- Visual Build provides additional options for handling nesting of build rules and checked status. To match the behavior of Visual Build, the options to nest build rules, nest include in build status, and

re-evaluate build rules should be unchecked.

- Since global macros are no longer stored in the registry, setting of global macros by modifying the registry is no longer supported. Script code that does this should be replaced by script expressions or steps with a Set Macro action or a Run Script action that manipulates the Application.Macros objects programmatically via the automation object model.
- Literal parenthesis and comma characters can now be inserted within macro parameters by using a double parenthesis (( )) or comma character ,,. Existing projects that include single literal parenthesis characters within macro parameters will need to be changed to use double parenthesis characters. For instance, %MYMACRO(some text( something else)% would need to be changed to %MYMACRO(some text(( something else)%). This only affects references to macros that accept parameters.

### Other changes to be aware of

- If Visual Build 3 is installed, when Visual Build Pro is first started, all the Visual Build global macros (stored in the registry) will be loaded into the Visual Build Pro global macros.
- Projects created in versions prior to version Visual Build 3 must first be opened and saved in Visual Build 3 before they can be opened in Visual Build Pro 4+.
- Bracket characters within fields now denote script code; to enter literal bracket characters, use double brackets ([[, ]]).
- The following conversions are performed when a project from a previous version is opened:
  1. Steps with no command or start in path are converted to Group actions.
  2. Steps using the SET\_TEMP\_MACRO or SET\_GLOBAL\_MACRO macros are converted to Set Macro actions.
  3. Steps using the LOGMSG macro are converted to Log Message actions.
  4. Step-specific system macros (STEP\_NAME, etc.) are converted to the equivalent script code that uses the Visual Build object model.
  5. Single bracket chars are converted to double brackets since brackets now denote script code.
- If the following macros are used in a project, do not uninstall Visual Build 3. If Visual Build 3 is installed, those macros will also be defined, and will operate correctly, in Visual Build. If and when those steps are converted to native Visual Build actions, Visual Build Pro3 will no longer be required:

<b>Macro</b>	<b>Visual Build Pro Equivalent</b>
PROCMASK	Process Files/Run Program actions
SENDMAIL	Send Mail action
REGSVR	COM Register action
REGSVREX	Process Files/COM Register actions
REG_WRITE	Write Registry action
ECHOW/ECHOA	Write File action
VSMAKE	Make VB6/Make VC6/VS6 Get Version actions
WISE	Wise Setup action

- For the Run Program action, the Ignore Fail property now will also ignore a failure to start a process. Previously, if the process for a command failed to start, the build would fail even if it was marked to ignore failure.
- The following options are no longer available:
  1. No option to determine whether or not to prompt for undefined macros in build rules (will prompt in GUI app unless /s switch was specified on the command-line, will not prompt in console app).
  2. The /a command-line switch is no longer supported.

3. No step-specific macros (STEPNAME, STEPOUTFILE, etc.). Use the equivalent script code ([Step.Name], [Step.OutputFile], etc.).
4. Option to prevent LOGFILE macro from being overridden was removed.

### 1.6.2 Visual Build Pro 4

- Existing projects created in Visual Build Pro 4 can be loaded by Visual Build 5+. The file schema has not changed, but if steps for new custom actions are added to the project or new features are utilized, projects saved in the new version may not build correctly in earlier versions.
- The default location for application configuration files has changed. The default path for version 4 was the application installation path. For backward compatibility, if existing configuration files exist in the application installation directory, they will be copied to the new default path during installation. This change was made because not all users have the necessary rights to modify files in the installation path.
- In version 4, the object model Macros.Load documentation was incorrect (the logic for the Clear parameter was backwards). In Version 5, the default has been changed from True to False to match the actual behavior of prior versions when a value was not passed.
- The LOGFILE system macro, configured in Application options, is no longer automatically enclosed in double quotes. If the LOGFILE macro contains spaces and is passed as a command-line argument in a Run Program step, use the QUOTE\_STR system macro to add quotes if necessary (i.e., %QUOTE\_STR(%LOGFILE%)%).
- Version 5 optimizes the size of project files when a step is edited by removing step properties that 1) match their default values and 2) the property default value is the default for that field type (unchecked for checkboxes, first item in a drop-down list combo, or an empty string for string fields). This will reduce the size of projects in most cases, but do not be alarmed if some properties are removed from a step after it has been edited and the project saved.
- The Scripts tab has been removed from the project view. Scripts can still be accessed and managed from the Script Editor, which is available from *View | Other Windows | Script Editor* on the menu bar.
- Macros are no longer allowed to contain spaces in the name. This was previously allowed, but an error would occur if the macro was referenced and expanded during a build. Existing projects with such macros can be loaded, but it is not allowed on new macros or when renaming existing macros.

### 1.6.3 Visual Build Pro 5

- Existing projects created in Visual Build Pro 5 (and earlier versions) can be loaded by Visual Build 6. The file schema has changed, so projects saved in the new version will not load or build correctly in earlier versions.
- Visual Build Pro version 6 can be installed side-by-side with previous versions. The default installation path is `C:\Program Files\VisBuildPro6` rather than `C:\Program Files\VisBuildPro`. Scheduled tasks, command (.cmd) scripts, batch (.bat) files, and Windows shortcuts that specify a full path for launching Visual Build will need to be adjusted for the new installation path (operations that do not specify a path [i.e., `start VisBuildPro`, double-clicking a .bld file, etc.] will launch the most recently installed version of Visual Build Pro).
- The default application data folder is `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 6` rather than `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional`. During installation, any configuration files in the

version 4/5 application data folder or install path will be copied to the version 6 folder. Also, the registry key to override the default application data path is `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 6\ConfigFilePath` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional\ConfigFilePath`.

Additional command-line options for specifying the location of application configuration files are available.

- Several obsolete system script functions are no longer included in system scripts. Either convert your projects to use the replacement actions documented here, or copy the script code from that topic into your global script code.
- The following obsolete global macros will be copied from an earlier installation but are no longer automatically created: WINZIP, VB, MSDEV, DEVSTUDIODIR, DEVENV. Replace their use with an equivalent built-in action or create and initialize as appropriate if needed on a clean install.
 

Macro	Visual Build Pro 6+ Equivalent
WINZIP	UNZIP Files/ZIP Files actions
VB	Make VB6 action
MSDEV	Make VC6 action
- The build status of the Run Program action will always be a value from `BuildStatusEnum` and not the exit code of the process. The exit code is available in the `RUNPROGRAM_EXITCODE` temporary macro.
- For performance reasons, v6.3 introduced changes to the default threading model of the Builder component and custom actions.
- All Visual Build COM components, type libraries, ProgIDs, and interface GUIDs are different in version 6. User action components will need to be updated to reference the Visual Build 6 Server type library and recompiled in order to work with version 6. Script code that uses ProgIDs of `"VisBuildSvr.Application"` and/or `"VisBuildSvr.Builder"` will need to be modified to use `"VisBuildSvr6.Application"` and/or `"VisBuildSvr6.Builder"` instead.
- Actions are now registered under `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 6\Actions` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional\Actions`. The COM Register action in version 6 registers actions at this location.
- User actions are now always identified by their filename (without extension). In previous versions, the filename was used by default, but this could be overridden by adding a Name attribute to the action element of the `.action` file. In version 6, the Name attribute is ignored and the filename is always used for the action name. To convert existing actions that have a Name attribute which differs from the filename, use the `ConvertActions.bld` project (located in the `Samples\User Actions` folder within the Visual Build installation path). This project processes all `.action` files in the path where it is located to identify actions whose Name does not match its filename. Then it will update all `.bld` files in the same directory from the old to the new action name. To process `.action` and/or `.bld` files elsewhere, change the `%PROJDIR%` value in the `Process.action files` and `Process.bld files` steps (and check the recurse flag to process subdirectories if desired).
- Custom logging components are now registered under `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 6\Logging` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional\Logging`.

## 1.6.4 Visual Build Pro 6

- Existing projects created in Visual Build Pro 6 (and earlier versions) can be loaded by Visual Build 7. The file schema has changed, so projects saved in the new version will not load or build correctly in earlier versions.
- Visual Build Pro version 7 can be installed side-by-side with previous versions. The default installation path is `C:\Program Files\VisBuildPro7` or `C:\Program Files (x86)\VisBuildPro7` (on 64-bit Windows) rather than `C:\Program Files\VisBuildPro6`. Scheduled tasks, command (.cmd) scripts, batch (.bat) files, and Windows shortcuts that specify a full path for launching Visual Build will need to be adjusted for the new installation path (operations that do not specify a path [i.e., `start VisBuildPro`, double-clicking a .bld file, etc.] will launch the most recently installed version of Visual Build Pro).
- The default application data folder is `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 7` rather than `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 6`. During installation, any configuration files in the version 4/5/6 application data folder or install path will be copied to the version 7 folder. Also, the registry key to override the default application data path is `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 7\ConfigFilePath` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 6\ConfigFilePath`.
- `ListFiles.vbs` from earlier versions is no longer installed in the `TOOLSDIR`. Use the List Files action instead or copy it from an earlier version.
- The following obsolete global macros will be copied from an earlier installation but are no longer automatically created. Replace their use with an equivalent built-in action or create and initialize as appropriate if needed on a clean install.

<u>Macro</u>	<u>v7 Equivalent</u>	<u>Macro Value</u>	<u>Parameters</u>
ATTRIB	Set File Attributes	%DOSCMD% attrib FLAGS "FILENAME"	FILENAME, FLAGS
BCB	Make C++Builder	"%BCBDIR%\Bin\BCC32.exe"	
BCBDIR		C:\Program Files\Borland\CBuilder5	
BJB	Make JBuilder	"%BJBDIR%\Bin\JBuilder.exe" -build	
BJBDIR		C:\JBuilderX	
CS_NET	Make VS.NET	"%DOTNET_DIR%\csc.exe"	
DELPHI	Make Delphi	"%DELPHIDIR%\Bin\DCC32.exe"	
DELPHIDIR		C:\Program Files\Borland\BDS\4.0\	
DEVENV_NET	Make VS.NET	"%DEVSTUDIO_NET_DIR%\Common7\IDE\devenv.com"	
REGEDIT	Write Registry	RegEdit /s "FILENAME"	FILENAME
VB_NET	Make VS.NET	"%DOTNET_DIR%\vbc.exe"	

- All Run Program-derived actions have been enhanced with Advanced and Remote tabs. The *Log the command-line used* option (typically on the Options tab in earlier versions) has been moved to the Advanced tab.
- All Visual Build COM components, type libraries, ProgIDs, and interface GUIDs are different in version 7. User action components will need to be updated to reference the *Visual Build Professional 7 Server* type library and recompiled in order to work with version 7. Script code that uses ProgIDs of "VisBuildSvr6.Application" and/or "VisBuildSvr6.Builder" will need to be modified to use "VisBuildSvr7.Application" and/or "VisBuildSvr7.Builder" instead.



- Actions are now registered under `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 7\Actions` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional\Actions`. The COM Register action in version 7 registers actions at this location.
- Custom logging components are now registered under `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 7\Logging` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 6\Logging`.

### 1.6.5 Visual Build Pro 7

- Existing projects created in Visual Build v4 and later can be loaded by Visual Build v8. The file schema has changed, so projects saved in the new version will not load or build correctly in earlier versions. Project files created with Visual Build v3 and earlier should be opened and saved in Visual Build Pro v7 before opening in v8.
- Visual Build v8 does not support Windows 2000. Windows XP or later is required.
- Visual Build Pro version 8 can be installed side-by-side with previous versions. The default installation path is `C:\Program Files\VisBuildPro8` rather than `C:\Program Files\VisBuildPro7`. Scheduled tasks, command (.cmd) scripts, batch (.bat) files, and Windows shortcuts that specify a full path for launching Visual Build will need to be adjusted for the new installation path (operations that do not specify a path [i.e., start VisBuildPro, double-clicking a .bld file, etc.] will launch the most recently installed version of Visual Build Pro).
- The default application data folder is `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 8` rather than `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 7`. During installation, any configuration files in the version 4/5/6/7 application data folder or install path will be copied to the version 8 folder. Also, the registry key to override the default application data path is `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 8\ConfigFilePath` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 7\ConfigFilePath`.
- Some obsolete actions are no longer displayed by default (existing steps for these actions will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- In Visual Build v8, the Sign Code action calls `signtool.exe` for signing. The action will attempt to automatically locate `signtool.exe` in a Microsoft Windows SDK installation or the Visual Build Tools path. If not found, either install the Tools option of the Microsoft Windows SDK or reinstall Visual Build and select *Full installation*.
- When opening projects in the 64-bit edition of Visual Build, ZIP Files and UNZIP Files actions will be converted to Enhanced Zip Files and Enhanced Unzip Files actions if possible, since the older compression actions are not available in the 64-bit edition. The conversion is only performed for steps that do not use options available only in the older actions, such as TAR/GZ compression, short filenames, or clearing of the archive flag. Steps that use these options will need to be replaced manually in order to build properly in the 64-bit edition.
- All Visual Build COM components, type libraries, ProgIDs, and interface GUIDs are different in version 8. User action components will need to be updated to reference the *Visual Build*

*Professional 8 Server* type library and recompiled in order to work with version 8. Script code that uses ProgIDs of "VisBuildSvr7.Application" and/or "VisBuildSvr7.Builder" will need to be modified to use "VisBuildSvr8.Application" and/or "VisBuildSvr8.Builder" instead (or the version-independent ProgIDs "VisBuildSvr.Application" and "VisBuildSvr.Builder").

- In previous versions, all script code used the US English locale. In v8, all script code uses the currently configured Windows locale by default. This can affect display of error messages, formatting and conversion of date/time values, etc. To revert to the old behavior, add a Run Script step to the beginning of the project with the code `Application.Options.UseUSEngishLocaleForScriptEngine = True.`
- Several obsolete system script functions are no longer included in system scripts. Either convert your projects to use the replacement actions documented here, or copy the script code from that topic into your global script code.
- The following actions no longer use the DOTNET\_DIR and DOTNETSDK\_DIR macros to locate the .NET command-line executables. Instead, the actions look for the latest installed .NET Framework or SDK path, or the path+filename of the executable to call can be specified in the Override field on the action's Options tab: COM Register, Export Type Library, GAC Install, Generate Resource Files, Import Type Library, Install .NET Services, PEVerify, Strong Name Tool, WSDL, XSD.
- The SSHELPER tool is no longer installed with Visual Build 8. Either replace calls to this tool with the SourceSafe action or download it from [here](#).
- Actions are now registered under `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 8\Actions` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional\Actions`. The COM Register action in version 8 registers actions at this location.
- Custom logging components are now registered under `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 8\Logging` rather than `HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 7\Logging`.

### 1.6.6 Visual Build Pro 8

- Existing projects created in Visual Build Pro v4 and later can be loaded by Visual Build Pro v9. Projects saved in the new version will not load or build correctly in earlier versions.
- Visual Build v9 does not support Windows XP. Windows 2003 or later is required.
- Visual Build Pro version 9 can be installed side-by-side with previous versions. The default installation path is `C:\Program Files\VisBuildPro9` rather than `C:\Program Files\VisBuildPro9`. Scheduled tasks, command (.cmd) scripts, batch (.bat) files, and Windows shortcuts that specify a full path for launching Visual Build will need to be adjusted for the new installation path (operations that do not specify a path [i.e., start VisBuildPro, double-clicking a .bld file, etc.] will launch the most recently installed version of Visual Build Pro).
- The default application data folder is `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 9` rather than `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 8`. During installation, any configuration files in the version 4 thru 8 application data folder or install path will be copied to the version 9 folder. Also, the registry key to override the default application data path is

HKEY\_LOCAL\_MACHINE\Software\Kinook Software\Visual Build Professional 9  
\ConfigFilesPath rather than HKEY\_LOCAL\_MACHINE\Software\Kinook  
Software\Visual Build Professional 8\ConfigFilesPath.

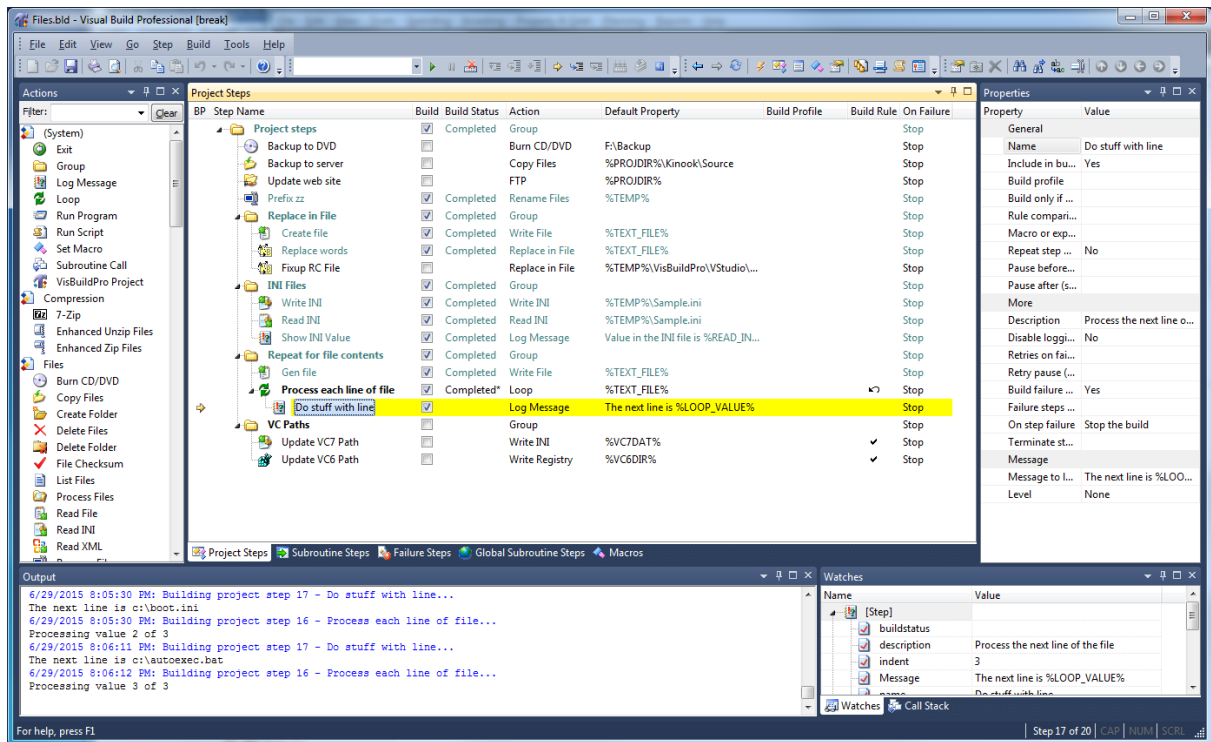
- Some obsolete actions are no longer displayed by default (existing steps for these actions will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- All Visual Build COM components, type libraries, ProgIDs, and interface GUIDs are different in version 9. User action components will need to be updated to reference the *Visual Build Professional 9 Server* type library and recompiled in order to work with version 9. Script code that uses ProgIDs of "VisBuildSvr8.Application" and/or "VisBuildSvr8.Builder" will need to be modified to use "VisBuildSvr9.Application" and/or "VisBuildSvr9.Builder" instead (or the version-independent ProgIDs "VisBuildSvr.Application" and "VisBuildSvr.Builder").
- Actions are now registered under HKEY\_LOCAL\_MACHINE\Software\Kinook Software\Visual Build Professional 9\Actions rather than HKEY\_LOCAL\_MACHINE\Software\Kinook Software\Visual Build Professional 8\Actions. The COM Register action in version 8 registers actions at this location.
- Custom logging components are now registered under HKEY\_LOCAL\_MACHINE\Software\Kinook Software\Visual Build Professional 9\Logging rather than HKEY\_LOCAL\_MACHINE\Software\Kinook Software\Visual Build Professional 8\Logging.

## 2 Using Visual Build

### 2.1 Main Screen

The main Visual Build screen contains a menu bar, several toolbars, an actions pane for inserting new steps, several step panes, a macros pane, a properties pane, an output pane, watches pane, and call stack pane, all of which can be customized in various ways. To get started, check out the samples included with Visual Build, and then add the steps, macros, and script that make up your project. You can easily debug the project step by step as you construct it.

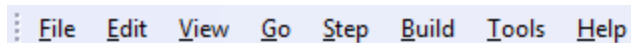
Click on the different areas of the image below to see more details.



Visual Build projects are stored in files with a `.bld` or `.bldx` extension. By default, Visual Build opens the most recently used project at startup, although this can be disabled via a user option or by holding Shift down when starting. Visual Build is an SDI app (only one file can be open at a time), but multiple instances of Visual Build can be started.

## 2.2 Menu Bar

The menu bar provides access to all application features. The menu is accessible via the mouse (by clicking on a menu item) or keyboard (by typing Alt+ the letter of a top-level menu item). Keyboard shortcuts are also predefined for many commands and can be customized by the user.



The menu bar can be repositioned (docked) anywhere around perimeter of the application and can also be undocked and moved anywhere on the screen. The menu bar can also be customized as desired.

## 2.3 Toolbars

Several toolbars are displayed by default. Each toolbar displays buttons for invoking application commands. To hide or show a toolbar, right-click on the menu bar, then click the toolbar to be shown or hidden. Alternatively, use the Customize Dialog to hide and show toolbars.



## Profiles

The **Build** toolbar contains a drop-down field for entering a build profile to specify which sets of steps to build (the `_BUILD_PROFILE_` temporary macro will be updated to this value when building). Type in or select a value in the list and press Enter to set it. Only steps with a profile matching the overall

build profile will be included in the build. The build profile can be a literal string or a regular expression (for instance, *Debug.\** to match all step profiles beginning with Debug, *.\** to match all, or *Release|Debug* to match either the Debug or Release profile). Comparisons are not case sensitive. The menu item *Build | Profile* and related shortcut can also be used to set focus to this field. The build profile can also be defined on the command-line when calling a project.

## Positioning

Toolbars can be repositioned (docked) around the entire perimeter of the application and also undocked and placed anywhere on the screen:

- If currently docked, drag the toolbar from the toolbar handle at the left-edge
- If currently undocked, drag the toolbar by its upper blue caption bar

Notes:

- If you drop the toolbar around the perimeter of the application, the toolbar will become docked.
- The outline that is displayed as you drag indicates where the toolbar will dock when the mouse is released.

## Customizing

To add, remove, or reset the buttons displayed for toolbars, click the Down Arrow button at the right end of each toolbar. Buttons can also be added or removed while the Customize Dialog is displayed.

Creating a new toolbar:

1. Display the Customize Dialog.
2. On the Toolbars tab, click the New... button.
3. Enter an appropriate name for the new toolbar, then click OK.
4. Add buttons to this new toolbar.
5. Close the dialog.
6. Reposition your new toolbar as desired.

To delete a toolbar, click the Delete button (if it is disabled, then the selected toolbar is a system toolbar and can't be deleted). To reset a toolbar to its default settings, click the Reset button.

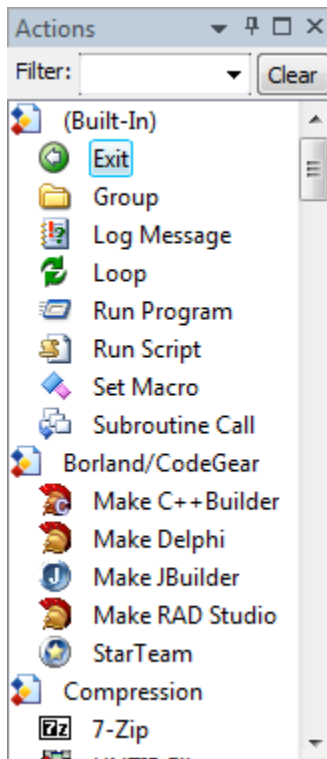
## Appearance

Several options related to toolbars and the menu bar can be customized as well. These options are viewed and changed on the Options tab of the Customize Dialog.

## 2.4 Panes

### 2.4.1 Actions Pane

This pane can be used to select an action for inserting a new step into a project or for creating and managing user actions. It is accessed by choosing *Edit | Insert* or *View | Actions* on the menu bar.



**Filter:** The Filter field can be used to quickly filter the list to locate a specific action or category. As text is entered, the action list is filtered to match only those actions or categories containing the typed text. The up and down arrow keys can be used to navigate in the list while focus is in the Filter field.

*Note:* Individual actions and categories that are not used can also be hidden from the list by right-clicking on the action and choosing *Hide* (use *Show Hidden* to show all hidden actions) or via the View menu.

**Clear:** Clears the Filter field and repopulates the action list with all registered, unhidden actions.

**Action list:** A categorized list of available custom actions that have been registered and are not hidden.

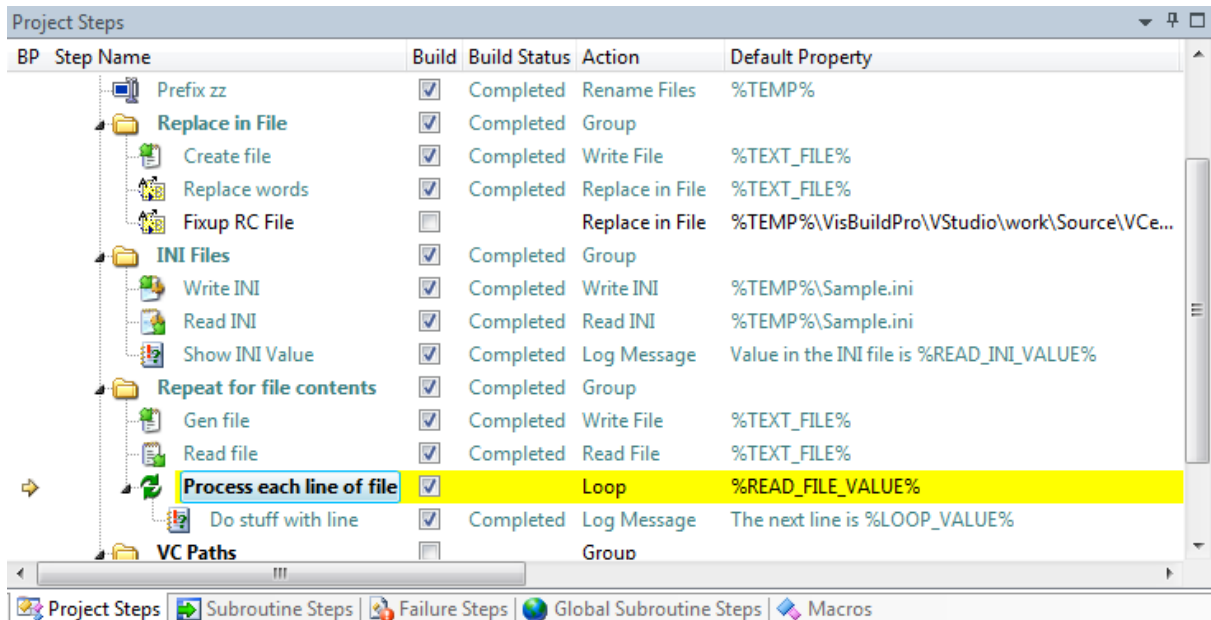
#### Creating a Step

- Double-click on a step or press Enter to insert a new step for the selected action into the current step pane.
- An action can also be dragged and dropped or copied and pasted onto a step pane to create a new step for the action.

### 2.4.2 Step Panes

By default, the Step Panes are docked in the upper half of the Visual Build window, with the following panes:

- Project Steps
- Subroutine Steps
- Failure Steps
- Global Subroutine Steps
- Macros



To switch between step panes, select the desired pane from the View menu, use the pane's shortcut key, or press the associated shortcut to cycle between panes. Panes can also be docked and customized in various ways.

Each pane displays a grid containing the information for that pane. Settings affecting how the grids are displayed can be configured in the related Options dialog. The columns can be resized by dragging the right edge of a column header. To hide a column, resize it so that it has no width, or right-click the column header to display a list of all columns to be shown or hidden. Columns can be rearranged by dragging and dropping the column headers as desired. The columns to be displayed on the step panes can also be configured in the associated options dialog. When the mouse cursor is held over a column (property), its expanded value will be shown in a tool tip.

*Note:* For actions whose default property is a filename or folder, right-click on the step and choose an item on the *Shell* to quickly open, explore, edit, or process the file or folder in a command prompt.

#### 2.4.2.1 Project Steps Pane

The Project Steps pane contains the main steps that comprise a project. Use this pane to add all the steps that will be performed during a build.

To add a new step, double-click an action on the Actions pane (use Insert on the Edit menu or press the Insert key to activate the Actions pane via the keyboard). The step name can be edited in place by clicking twice on its label. A check box column such as the Build column can be toggled by clicking on its check box. To edit all properties for an existing step, choose the Properties button on the toolbar, the Properties Dialog or Properties Window item on the View menu, or press Enter.

Each step pane contains all the steps of that type. Steps can be organized or grouped by indenting child steps below the parent steps by selecting one or more steps and choosing Move Left or Move Right from the Step menu. Steps can be rearranged by left or right-dragging and dropping them to a new location, by using the Step menu items, or by using the associated keystrokes. Double-click on a step or macro name to view its properties. Click on the Build checkbox or the step's icon (or press the Space bar) to toggle inclusion of that step in the build (checking/unchecking a step also toggles the checkbox for any of any child steps unless the Shift key is held down when clicking). Click in the BP column for a step to toggle a breakpoint for a step.

To quickly go to the build output for a step, choose *Go To | Step Output* from the menu bar. If a step's default property is a filename or folder, it can be quickly opened in the associated application (Launch), viewed in the user-configured viewer (View), or opened in Explorer (Explore) by choosing Launch, View, or Explore from the View|Shell menus or pressing the associated shortcut key.

For complex projects, a step with child steps can be expanded or contracted to show or hide the child steps. This is done by clicking on the plus or minus sign next to the step name, by choosing the menu item on the Step menu, or by pressing the left or right arrow key.

#### 2.4.2.2 Subroutine Steps Pane

The Subroutine Steps pane is where project subroutines are defined. Subroutines can be very useful for modularizing the code in your project. If there is a common set of steps that needs to be used in multiple locations in your build, you can place them in a subroutine and call them from several places in the project.

A subroutine is defined as any step (other than a Subroutine Call action step) at the main level (no indentation) of the Subroutine Steps pane, and it is comprised of that step and any child steps (indented below that step). Each subroutine should be given a unique name; if two subroutines with the same name are defined, the first one will be called.

To call a subroutine, add a step with the Subroutine Call action to your project. The custom action screen for this action allows the selection or entry of a subroutine step name and temporary macros to be defined before the subroutine steps are called (these macros values will also be deleted or their original value restored when the subroutine call returns). Subroutine steps can reference any available macro name, not just those defined in this list, but this provides a convenient way to quickly define any macros to pass to the subroutine steps.

Subroutine steps can be called from Project steps, Failure steps, child Subroutine steps, or Global Subroutines. Project subroutines will always be searched first for a matching subroutine name, and if not found, the global subroutine steps will be searched (this allows global subroutines to be overridden by a project subroutine of the same name). When a subroutine returns, all the temporary macros defined for calling that subroutine will be deleted, so be sure to use unique parameter names across subroutines when one subroutine calls another subroutine.

#### 2.4.2.3 Failure Steps Pane

The Failure Steps pane is used to define steps that should be executed when a step in the build fails. Multiple failure steps can be defined, and each step can be configured to build all or a certain set of failure steps upon failure. Information about the failed step is available in the FailedStep object and the FAIL\_STEP\_STATUS and FAIL\_STEP\_OUTPUT system macros. If a failure step fails and is not marked to continue, the build stops without processing any more failure steps.

#### 2.4.2.4 Global Subroutine Steps Pane

The Global Subroutine Steps pane is where global subroutines are defined. Global subroutines can be very useful for modularizing code across multiple projects. If there are common steps that needs to be used across multiple projects, you can place them in a global subroutine and call them from each project. By default, global subroutine steps are loaded/saved from/to the `VisBuildPro.steps` file in the configuration files path. Global subroutine steps are saved to disk whenever a project is saved or the GUI application is closed.

A subroutine is defined as any step (other than a Subroutine Call action step) at the main level (no indentation) of the Global Subroutines pane, and it is comprised of that step and any child steps (indented below that step). Each subroutine should be given a unique name; if two subroutines with the same name are defined, the first one will be called.

To call a subroutine, add a step with the Subroutine Call action to your project. The custom action screen for this action allows the selection or entry of a subroutine step name, and the entry of up to

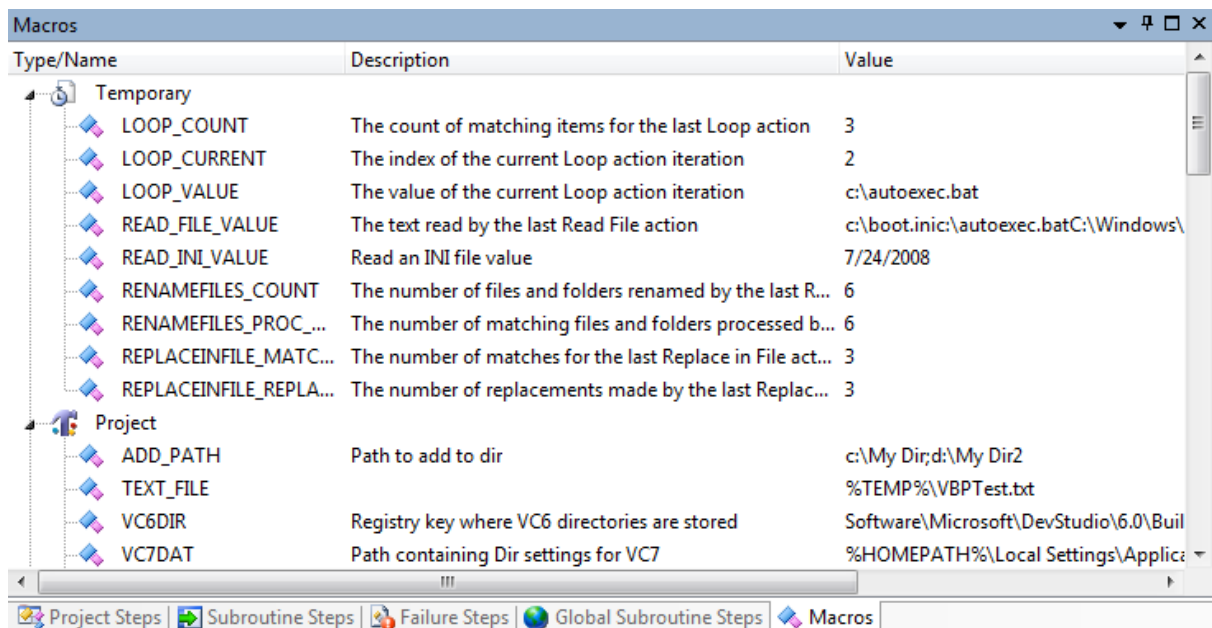


several temporary macros to be defined and passed to the subroutine. Subroutine steps can reference any available macro name, not just those defined in this list, but this provides a convenient way to quickly define any macros to pass to the subroutine steps.

Subroutine steps can be called from Project steps, Failure steps, or project Subroutine steps, or another Global subroutine. Project subroutines will always be searched first for a matching subroutine name, and if not found the global subroutines will be searched (this allows global subroutines to be overridden by a project subroutine of the same name). When a subroutine returns, all the temporary macros defined for calling that subroutine will be deleted, so be sure to use unique parameter names across subroutines when one subroutine calls another subroutine.

### 2.4.3 Macros Pane

The Macros pane is used to manage Visual Build macros. It is accessed by clicking on the Macros pane in the Step Panes, or by choosing the Macros item on the View menu. Macros provide a powerful capability that can be used to make projects more dynamic and generic.



There are four (4) types of macros:

**Temporary macros** are macros that exist only for the current Visual Build instance. They are assigned by passing in a VisBuildPro Project action or on the command-line, by inserting under the Temporary node above, or by creating them during a build in Set Macro step actions. These can be useful for temporary values that are not needed beyond the processing of a build, or to pass in dynamic values used in a build.

**Project macros** are saved with the project file.

**Global macros** are global across all projects and are stored in `VisBuildPro.macros` file in the configuration files path. Visual Build predefines several global macros, and additional global macros can be created here or within a build using the Set Macro action. Changes to global macros are saved before each step is built and reloaded (if changed externally) after a step is built and can be used to transfer values between projects.

**System macros** are the Windows environment variables plus several built-in macros and tools. System macros cannot be changed by the user.

Macros are referenced within properties of steps or macros using a format of %MACRO\_NAME%. If the same macro exists for more than one type of macro, the following order of precedence (from highest to lowest) determines which macro is used:

- Temporary
- Project
- Global
- System (Environment variables take precedence over predefined Visual Build system macros)

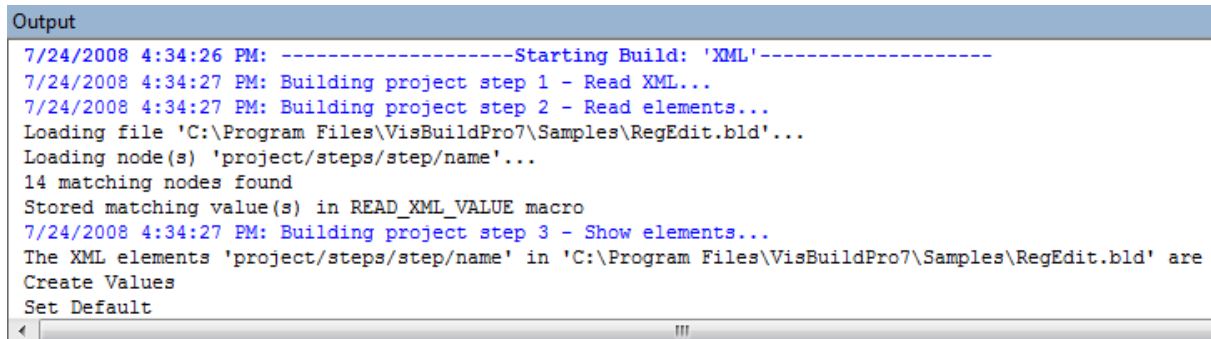
Macro overrides can be very useful; for instance, a project macro could be defined as a default value, but a temporary macro passed on the command-line when building the project. This temporary macro would override the value defined for that project macro.

Macros can be cut, copied and pasted from within the Macro tab and from one Visual Build instance to another. A macro can be renamed by clicking twice on its name or pressing F2. Macro names are not case sensitive. When the mouse cursor is held over the Value column, its expanded value will be shown in a tool tip.

The Macros pane is updated when the build pauses or stops at a breakpoint and can be manually refreshed by choosing *View | Refresh* on the menu.

#### 2.4.4 Output Pane

The Output pane is displayed in the lower half of the Visual Build window. The Output pane contains a colored edit control which displays all build output.



```
Output
7/24/2008 4:34:26 PM: -----Starting Build: 'XML'-----
7/24/2008 4:34:27 PM: Building project step 1 - Read XML...
7/24/2008 4:34:27 PM: Building project step 2 - Read elements...
Loading file 'C:\Program Files\VisBuildPro7\Samples\RegEdit.bld'...
Loading node(s) 'project/steps/step/name'...
14 matching nodes found
Stored matching value(s) in READ_XML_VALUE macro
7/24/2008 4:34:27 PM: Building project step 3 - Show elements...
The XML elements 'project/steps/step/name' in 'C:\Program Files\VisBuildPro7\Samples\RegEdit.bld' are
Create Values
Set Default
```

All build output is logged to the Output pane (and a log file if enabled), depending on the configured log level. Each step that is built is logged, followed by any step output, and the status of the step. The beginning and completion of a build is also logged. The colors and other output settings can be configured via the Edit menu and in the Colors, Output, and Build options dialogs.

Several capabilities are available for quickly navigating through the build output.

#### 2.4.5 Properties Pane

The Properties pane/window can be used to view and edit step and macro properties. It is accessed by clicking on the Properties pane or by choosing the *Properties Window* item on the View menu. This is an alternative, modeless pane for modifying properties. The modal step and macro properties dialogs can also be used to edit properties. A user option determines which method is used by default when inserting new steps and macros.

Property	Value
Pause before (seconds)	
Pause after (seconds)	
More	
Description	Fixup all dialog resources to...
Disable logging of action...	No
Retries on failure	
Retry pause (seconds)	
Build failure steps	Yes
Failure steps to build	
On step failure	Stop the build
Terminate step after (sec...	
Replace	
Input file	%TEMP%\VisBuildPro\VStu...
Output file	
Ensure output file is writ...	Yes
Perform case-sensitive ...	Yes
Replace first match only	No
Treat all characters as lite...	No
Dot does not match new...	No
Allow comments and ig...	No
Binary mode	No
Log file contents before ...	No
Do not replace, only log ...	No
Encoding	
Text	
Text or regular expressio...	(^STYLE\s+WS_([[^\s]]*\s+...
Text or regular expressio...	(?1STYLE DS_SHELLFONT \   ...
Text to append to file if n...	

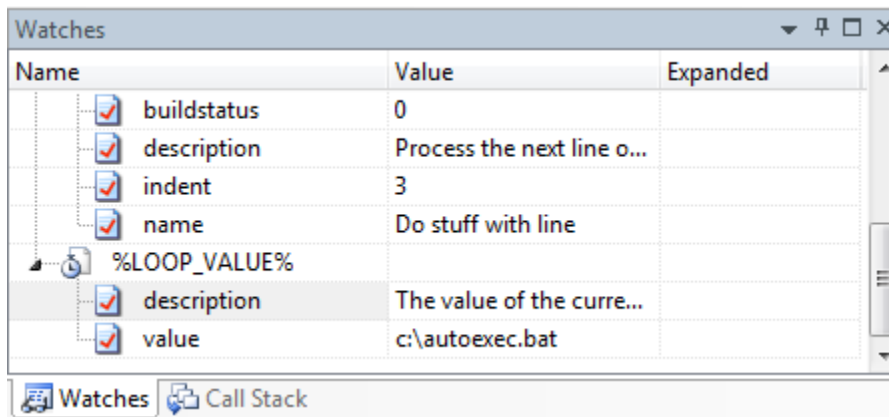
Press the arrow keys to move up or down row by row, or press Ctrl+Tab/Shift+Ctrl+Tab to go to the next/previous grouping/page. To edit a property value in the pane, double-click on the value, press F2, or begin typing a new value. To enter a macro reference, type % and additional characters to filter the macro list. Press Enter to accept the new value or Esc to cancel editing. For multiline, array and fields with override values, the step properties dialog will be opened to modify the property. The expanded value of a property will be displayed in a tool tip when hovering the mouse over that column.

Other functions are available via the context menu and keyboard shortcuts:

- Property values can be cut, copied, pasted.
- A property containing a filename or path can be opened a file in its associated application (Launch), viewed file in the user-configured viewer (View), or the folder opened in Windows Explorer (Explore). Any macros or script are expanded to determine the filename to be used.
- The related step or macro properties dialog can be opened to edit the properties.

## 2.4.6 Watches Pane

The Watches pane can be used to view step and macro properties and script expression values while building and editing a project. It is accessed by choosing *View | Other Windows | Watches* on the menu bar.



To add new watch items:

- Drag/drop or copy/paste steps from a Step pane into the Watches pane.
- Drag/drop or copy/paste macros from the Macros pane into the Watches pane.
- Open the Script Editor, enter a script expression in the Immediate field, then choose *File | Insert and Close* on the menu to add a watch for that expression.
- Focus the Watches pane, choose *Edit | Insert* on the menu, enter a partial macro name and OK the Insert Macro dialog to insert the selected macro.
- Focus the Watches pane, choose *Edit | Insert* on the menu, cancel the dialog, and type an macro name or script expression into the Watches pane.

To delete a watch item, select it and choose *Edit | Delete* on the menu.

Some common objects and macros that can be useful to watch:

- [Step]
- [LastStep]
- [FailedStep]
- LASTSTEP\_OUTPUT
- LASTSTEP\_STATUS

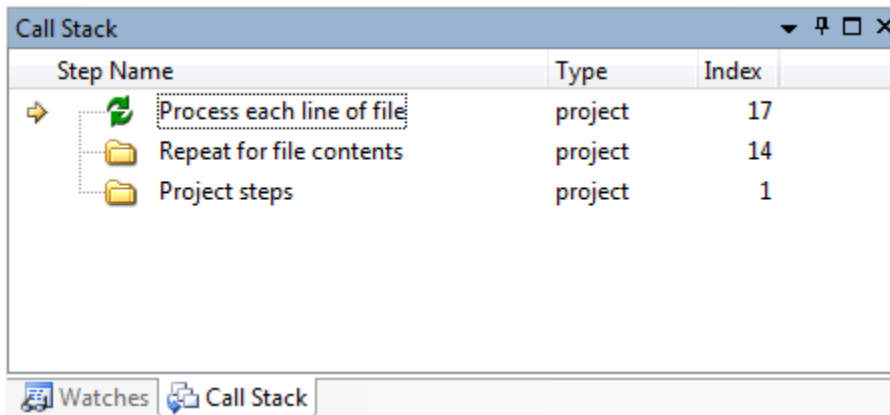
For expressions that evaluate to steps (i.e., [Step] or [FailedStep]) or macros (i.e., LASTSTEP\_OUTPUT, %WINDIR%, etc.), each step or macro property value will be shown in the Watches pane. For expressions that evaluate to arrays, each array element will be shown. The raw (unexpanded) value of each property will be displayed in the Value column, and the expanded value will be displayed in a tool tip when holding the mouse over the field.

For complex expressions (script expressions and multiple macro references), the expanded/evaluated value will be displayed in the Expanded column.

The Watches pane is refreshed when the build pauses or stops at a breakpoint and can be manually refreshed by choosing *View | Refresh* on the menu.

## 2.4.7 Call Stack Pane

The call stack pane can be used to view the build call stack while stepping through a build. It is accessed by choosing *View | Other Windows | Call Stack* on the menu bar.



The current step is displayed at the top, followed by each parent step on the build call stack. Double-click on a step (or select and press Enter) to navigate to that step. The call stack is automatically refreshed when the build pauses or stops at a breakpoint and can be manually refreshed by choosing *View | Refresh* on the menu.

## 2.4.8 Customizing Panes

The various application panes can be customized to suit your particular needs (similar to pane customization in Visual Studio):

1. Panes can be docked around the perimeter of the main window, docked together in a tabbed group, or undocked/floated and moved anywhere on the screen. Toggle the docked state of a pane by double-clicking the pane's caption bar, right-clicking the caption bar and choosing *Floating*, or dragging the pane by its caption. When a pane is undocked, it can be placed outside of the application window, otherwise it will be snapped to the main window. Dragging a pane will display an outline of the locating where it will be positioned when the pane is dropped. Drop onto the appropriate docking sticker button to dock a pane or add it to a tabbed group; drop somewhere other than on a docking sticker to undock a pane at the specified location.
2. Panes can be configured to automatically hide or "shrink" to the perimeter of the application until needed:
  - To toggle a docked pane's Auto hide mode, click its Auto Hide button or right-click the caption and choose *Auto hide*.
  - To display (expand) an auto-hidden pane, either hover the mouse over the tab for the pane, click the menu item for that pane, or type the keyboard shortcut for that pane.
  - To hide a pane that is in auto-hide mode, change the focus to a different pane.
3. Each pane can be resized using the splitters that are displayed around the perimeter of each pane (when docked). Panes can also be resized like a normal window when undocked.
4. Most panes can be closed by clicking the close button displayed in the upper right corner, by right-clicking the caption and choose *Close*. To display a closed pane, simply navigate to that pane using the Menu Bar or toolbar entry or press the related Keyboard Shortcut.

*Note:* Pane customizations are stored in the registry under the key `HKEY_CURRENT_USER\Software\Kinook Software\Visual Build Professional 9\Options\DockingPaneLayouts`. To reset all pane customizations to their defaults, exit Visual Build, download this file and run `ResetPanes.reg` by double-clicking (accepting all prompts).

## 2.5 Layouts

An arrangement of panes, menus, and toolbars is referred to as a layout. Two independent layouts are available and each can be configured separately. One layout is displayed during design mode

when editing a project, and another layout is displayed when building if the related user option is enabled.

## 2.6 Building Projects

The actual build process is where the power of Visual Build can be seen. The topics below describe the build process in detail:

- Definition of Terms
- Methods of Building
- Build Profiles
- Build Details
- Build Logging

### 2.6.1 Definition of Terms

The *current build position* is indicated by the yellow arrow in the far left column of the step grid. During a build, the entire current step row is also highlighted in yellow. The current build position is where a build will normally start or continue from. Build commands are also available that change the current build position before launching a build.

One or more steps can be selected in the step grid; *selected steps* are indicated by the step name being highlighted (or the entire row if the full row select option is enabled). The first selected step is also referred to as the *cursor position*.

Each step also has an *include in build flag* associated with it (indicated by a checkbox in the Build column of the Step Panes). A check mark in the Build column indicates that the step (and any child steps if the nesting option for *Include in Build* status is checked) will be included in the build.

Finally, each step has a *build status* displayed in the Build Status column, which can have one of the following values:

<u>Status</u>	<u>Meaning</u>
(blank)	The step has not been built
Building...	The step is currently being built
Failed	The step failed when built
Aborted	The build was canceled by the user while the step was building
Skipped	The step was skipped due to a build rule or unchecked step
Terminated	The step was terminated because it didn't complete in time
Completed*	The step was completed but may have additional iterations (for repeating build rules, Process Files action, or Loop action)
Completed	The step was successfully built

When a project is built, if a step is checked, has no (blank), Failed, or Aborted status, and the current position moves to that step, it will be considered for building. If the build profile matches and all applicable conditional build rules evaluate True, it will be built, otherwise it will be skipped. If a step has a status of Skipped or Completed, that step will not be considered for building again, unless its build status is reset by 1) unchecking and checking that step, 2) rebuilding the project, 3) resetting the build status for all steps, or 4) closing and reopening the project.

### 2.6.2 Methods of Building

The following methods are available for starting or continuing a build (shortcut keys are available for all commands)

<u>Command</u>	<u>Behavior</u>
Build	Starts from the current build position and executes all checked steps that have not been completed or skipped

Build From Cursor	Sets the current build position to the first selected step and performs a build
Build Step Group	Sets the current build position to the first selected step and builds that step and any uncompleted child steps, stopping when a step at the same indentation as the first step is reached
Build Next Step	Starts/continues the build from the current build position with single-step debugging, pausing before each step is built
Start From Cursor	Starts the build from the first selected step with single-step debugging, pausing before each step
Build To Cursor	Start/continue the build and break when the first selected step is reached
Rebuild	Clears the build status for all steps, sets the current build position to the beginning of the project, and performs a build
Rebuild Selected	Rebuilds all selected steps, regardless of any grouping, build status, conditional build rules, or include in build status*
Rebuild Step Group	Resets the build status of the selected step and children, sets the current build position to the first selected step and builds that step and any child steps
Restart	Resets the build status of all steps, starts a build and pauses at the first step
Reset	Resets the build status for all steps
Pause	Pauses the build after the current step completes
Stop	Terminates the currently building step and aborts the build (failure steps will also be built if the related option is enabled)

\*Note: Rebuild Selected is an ad hoc method to rebuild the selected steps regardless of step completion status, profiles, conditional build rules, and include in build status. *Iterating actions* (the Process Files action, Loop action, or a repeating build rule) are not compatible with rebuild selected or the Test button on the step properties dialog. Some ways to test build rules or iterating steps without building the entire project are:

- Select the iterating or parent step and choose *Build Step Group* or *Rebuild Step Group*.
- Select the iterating or first step, choose *Start From Cursor*, select the step following the last iterating child step, and choose *Build to Cursor*; then stop the build when it breaks on the selected step.
- Add a breakpoint following the iterating children, select the iterating or first step, and choose *Build From Cursor*.
- Put the iterating/parent + child steps in a subroutine (on the Subroutine Steps pane), add a Subroutine Call step in the project steps to call the subroutine, and *Rebuild Selected* can then be used on the Subroutine Call step.

Note: Breakpoints are stored in the project's user configuration file (`ProjectFile.bld.user`) in the same path as the project file.

### 2.6.3 Build Profiles

**Profiles** can be used to determine which sets of steps to build.

Each step can have a profile value, which is matched with the build profile. If both the build profile and step profile are empty, the step will be included in the build (if checked and build rules don't exclude it from building). If the build profile is not empty, only steps with an empty profile or a profile value that matches the build profile (see below) will be built. If a step with a non-empty profile does not match and is excluded from building, any child steps with an empty profile will also be excluded.

The build profile or the step profile can be a literal string or regular expression (for instance, *Debug.\** to match all step profiles beginning with *Debug*, *\** to match all, *Release/Debug* to match either the *Debug* or *Release* profile, etc.). If the build profile and step profile are identical, the step will be built. If the build profile is a regular expression, any step profile string that matches the expression will be built, or

if the step profile is a regular expression, any build profile value string that matches the step profile will result in the step being built. Comparisons are not case-sensitive. A single asterisk (\*) can also be used as a wildcard to match all profile values.

The **Build** toolbar contains a drop-down field for defining the build profile (which corresponds to the `_BUILD_PROFILE_` temporary macro). The build profile can also be defined on the command-line when calling a project.

## 2.6.4 Build Details

When a build starts, any existing macros that are marked to be added to environment variables are added. Any Set Macro steps in the build marked this way will also add/update or delete the associated environment variable for the Visual Build process when built.

### Build Status and Breakpoints

During a build, `[building]` will be displayed in the main window caption. If the build is paused at a breakpoint, because the build has been paused, or for single step debugging, `[break]` is displayed in the caption. Breakpoints can be set or cleared at any time by clicking on the BP column next to the desired step, selecting a step and choosing Step|Toggle Breakpoint. In the Step Panes, the current step is also highlighted, and *Building...* is displayed in the Build Status column. This view will automatically scroll down to show the current step. If the window is scrolled manually, automatic scrolling is halted.

*Note:* a step's build status is not saved with the project file unless the associated option is set.

### Steps Included in a Build

Each step is evaluated to determine whether it should be built. If the step is already marked *Completed* it will not be built again until its status is reset (by resetting the build or by unchecking and checking that step) or a Rebuild is performed. Next, the step and its parent steps must match all build conditions. If the evaluation of the checked status, profile, and build rules are successful, the step will be built; if the evaluation is false, the step will be skipped and its status updated to *Skipped*. When a step's repeating build rule is processed or when a step is skipped, if the logging option to log build rules when a step is skipped or repeated is checked, the details of the build rule will be logged.

*Note:* The profile, conditional build rules, and the checked flag are not considered when Rebuild Selected is chosen (all selected steps will be built).

### Current Build Position and Logging

The current build position is always marked with a yellow right arrow indicator in the leftmost grid column. If the build is paused at a step for debugging or due to a breakpoint, the entire row background will be highlighted. To activate the current step (for instance if it has scrolled out of view or is on a different tab), use the *Go | Current Step* menu item to scroll it into view and select it. The Call Stack pane can be used to view the stack while stepping through a build.

### Build Activity and Logging

All build activity is logged in the lower Output pane (and optionally a log file). Each step that is completed successfully is marked *Completed* in the Build Status column of the Step pane. For steps with repeating build rules or that use the Process Files or Loop actions, an asterisk will be displayed (Completed\*) to indicate that the rule will be reevaluated and may repeat or that there are more files to be processed. The Watches pane can be used to view step and macro properties and script expressions while building.



## Undefined Macros

If an undefined macro is encountered while building a step, Visual Build displays a dialog allowing the user to enter its value and choose which type of macro to create (project, global, or temporary). If the dialog is canceled, the macro remains undefined (and the step will fail unless it was marked to continue on failure).

## Build Error Handling

If an error occurs processing a step and the step is not marked to continue on failure, the build stops, the failed step is highlighted, and *Failed* is displayed in the Build Status column. Any error messages produced by the step and the process exit code are displayed in the build log.

If failure steps are defined (on the Failure Steps pane), when an error occurs building a step, the failure steps are executed before the build stops. This can be useful for sending e-mail as notification of the failure, undoing build operations, etc. If all failure steps complete successfully, the failed step in the Project or Subroutine Steps pane will be shown; if a failure step fails, the failure step will be highlighted.

View the build output to help determine why a step failed. To quickly go to the step in the Step Panes for a failed step, double-click on a line of the step's output. If a filename or folder is displayed in the build output, it can be quickly opened in the associated application (Launch), viewed in the user-configured viewer (View), or opened in Explorer (Explore) by clicking on that line or selecting the filename and choosing View from the context or View|Shell menu. Once the problem has been identified and corrected, choosing Build will continue the build from the failed step.

*Note:* Rebuild Selected is an ad hoc method for rebuilding the selected steps regardless of build rules, completion status, or include in build (checked) flag, and failure steps are not executed if a build fails during a Rebuild Selected. See the Methods of Building help topic for ways to test failure steps, build rules or iterating steps.

### 2.6.5 Build Logging

Build activity, including the output from each step, is logged in the Output pane or console and to a log file if file logging is enabled (depending on the log message level, logging options, and step output options).

The Output pane can be scrolled manually to view any output that has scrolled off the screen. The output for the selected step, or the step for selected output, can be quickly navigated to using the *Go / Step Output* menu item.

All build output can also be logged to a text or XML file. File logging can be enabled or disabled on a global, project, or build basis:

- **Global:** Configured by enabling logging in Application Options (creates a global LOGFILE macro).
- **Project:** Specified for a given project in the Project Properties dialog (creates a project LOGFILE macro; takes precedence over global settings for that project).
- **Build:** By passing in a temporary LOGFILE macro on the command-line (takes precedence over global or project settings for that run).

*Notes:*

- To disable logging, specify an empty string for the log filename.
- When logging to an XML file, the Transform XML Log action can be used to convert the log to an HTML report, RSS feed, etc.
- See the Logging.bld sample for more details.

## 2.7 Dialogs

This section explains the most common Visual Build dialogs in detail. Most of the dialogs in Visual Build are resizable, and their size and position are remembered from the last time the dialog was displayed.

### 2.7.1 About

The About dialog is accessed by choosing About from the Help menu. It displays program version, copyright, and license information.

The dialog contains a Register button for entering product registration information received after licensing Visual Build. When the evaluation period has expired, the product cannot be used until it has been licensed and registered. After purchasing, click the Register button and follow the directions for turning the evaluation version into a licensed version.

### 2.7.2 Action Properties

The Action Properties dialog is used to create, edit, or view the details of a Visual Build user action or system action. It is accessed from *Edit | Action Properties* on the menu bar. The dialog contains the following tabs:

- General
- GUI
- Component

*Note:* The properties of built-in actions (installed with Visual Build) can be viewed but not edited from this dialog.

#### 2.7.2.1 General

This tab of the Action Properties dialog is used to edit the general action properties of a user action. Fields on the dialog:

**Location:** The path of the `.action` file (read-only).

**Name:** The name of the user action (required). This is the name of the `.action` file without extension, and is the identifier of steps with this action type. Names should be chosen that are distinct from other existing actions.

**Save As:** Click this button to save the current action information under a different filename. Useful for using an existing action as the basis for another one.

**Description:** The description of the action (optional; shown in a tool tip in the Actions pane).

**Category:** The category the action will be listed under in the Actions pane (required).

**Default property:** The step property that will be displayed in a step grid's Default Property column (optional).

**Bitmap/Icon file:** The filename of a bitmap or icon file to use as the action's image in the Actions pane or step grid (optional). Relative filenames are relative to the `.action` file path. If not provided, a default image is used. For bitmaps, the top left pixel specifies the transparency color for the image.

**File extensions:** A list of file extensions that this action should be associated with (delimited by semicolons). When VisBuildPro.exe is started from the command-line with a filename of one of these extensions, a step with the related action will be created and populated with its filename (in its default

property). This can be useful for creating an Explorer context menu for the associated extension (i.e., by creating a registry value like `HKEY_CLASSES_ROOT\<extension name>\shell\Build` with `&Visual Build Pro\ = C:\Program Files\VisBuildPro9\VisBuildPro.exe "%1"`).

### 2.7.2.2 GUI

This tab of the Action Properties dialog is used to edit the general GUI tabs of a user action. Fields on the dialog:

**Tabs & Properties grid:** Configures the tabs and fields displayed in the Step Properties dialog in a step for the action.

*Note:* Not all columns are shown by default. To show additional columns, right-click the grid header and select a column name. Grid columns can also be resized by dragging the size bar or column header.

**Insert Tab:** Inserts a new Tab row into the grid (shortcut: Shift+Insert). Each tab row will display as another tab in the Step Properties dialog in a step for the action.

**Insert Property:** Inserts a new Property row into the grid (shortcut: Insert). Each property row will display a field on the Step Properties dialog. Any values that the user enters in these fields will be assigned to the step's properties and will be accessible from the action's component logic when the step is built. The following columns are available for each property:

<b>Name</b>	Defines the step property this field's data will be stored in (accessed in the action via <code>Step.ExpProperty</code> ). The value is case-sensitive. Standard step properties use all lowercase, so to avoid conflicts with the standard properties, use camel-cased property names (i.e., <code>PropertyXyz</code> ) for property names. For Grid fields, the value can either be a single property name for a single column grid or two property names separated by a semi-colon (;) character for a two-column grid for name/value pairs stored in separate step properties.
<b>Caption</b>	The caption that will be displayed next to the field on the step properties dialog. If left blank, the property name will be used instead.
<b>Type</b>	Defines the type of field to create. Supported types: <b>Edit:</b> an edit control with optional drop-down list <b>Check:</b> checkbox. <b>Combo:</b> a drop-down list combo <b>Grid:</b> a one- or two-column grid for entering an array of values <b>Radio:</b> a set of radio buttons
<b>Default Value</b>	Defines the default value to set the field to for new steps that are created (defaults to blank for Edit controls, unchecked for Check fields, the first item for Combo and Radio fields if not provided); the value should be a string value for Edit controls, 0 for unchecked/non-zero for checked Combos, and a 0-based index for Combo and Radio controls.
<b>Required</b>	Indicates whether the user will be required to enter a value for the field (applies only to Edit fields).
<b>Password</b>	Applies to Edit fields only; if checked, indicates that the value entered should be protected, displaying dots instead of the actual characters typed.
<b>Filter</b>	Specifies a filter to use when showing the Browse dialog in the Step Properties dialog. Separate the name and mask extensions and each mask with a vertical bar ( ), and separate each mask with a semicolon (;). For example: Text files (*.txt, *.csv) *.txt;*.csv Excel files (*.xls) *.xls
<b>Folder Browse</b>	If checked, the Step Properties dialog Browse button will display a folder browse dialog instead of a file browse dialog (the Filter field does not apply if this option is checked).
<b>List Data</b>	Defines a list of strings to add to a Radio, Combo, or Edit control field; separate list values with a semicolon (;) character. This field is required for Combos and Radio

	fields.
<b>Width</b>	Specifies a non-default width in dialog units for the control (the control is not automatically resized with the dialog if a width is specified).
<b>Height</b>	Specifies a non-default height in dialog units for the control.

**Delete:** Deletes the selected row from the grid (shortcut: Delete).

**Move Up:** Moves the selected row up in the grid (shortcut: Ctrl+U).

**Move Down:** Moves the selected row down in the grid (shortcut: Ctrl+D).

**Preview:** Displays the step properties dialog to show the layout of the GUI tabs and fields.

#### **Other shortcuts**

F2 or F4 or typing text: edit the selected cell's value.

Ctrl+C: Copy the selected row to the clipboard.

Ctrl+V: Paste the copied row, creating a new row.

### **2.7.2.3 Component**

This tab of the Action Properties dialog configures the actual logic of a user action.

**Action type:** Specifies how the action is implemented (either as script code or a COM component implementing the ICustomAction interface).

**Script language:** Specifies the script language for script actions. Use the Script Editor button to enter or edit the action script code that will be executed when a step for the action is built. The action code is implemented in a vbld\_BuildStep function, and the return value of this function is used to indicate the success/failure build status of the action. For VBScript, Jscript, and PerlScript languages, the vbld\_BuildStep method's signature will be created automatically; for other languages, create the method signature using the syntax for that language.

**ProgID:** The programmatic identifier for component actions.

**Runs program:** If checked, it indicates this is a Run Program-derived action. Two additional Advanced and Remote tabs will be displayed for the action in the Step Properties dialog, and these properties will be used when RunProgram or RunProgramEx is called by the action code.

**Temporary macros created by the action:** This defines a list of temporary macros (and their descriptions) that the action creates when built. The macros in this list will be included in the Insert Macro dialog.

### **2.7.3 Application Options**

This dialog is used to configure global application settings that are not user-specific but are global to all builds. It is accessed from *Tools | Application Options* on the menu bar. These options are also accessible via the object model's Options object.

- General
- Logging
- Logging (More)
- File Locations
- Advanced

By default, application options (except those on the Actions tab) are loaded and saved from/to `VisBuildPro.config` in the configuration files path.

### 2.7.3.1 General

The general tab of the application options dialog configures the following options:

**Default script language:** Specifies the language to use when executing script expressions within step and macro properties. Clear this field to disable scripting or if Windows Scripting is not installed.

**Set current directory to project folder after loading or saving projects:** If checked, whenever a project file is loaded or saved, the current directory for the process (`VisBuildCmd.exe` for the console app, `VisBuildPro.exe` for the GUI app, or the process that instantiated the object model) is updated to the folder containing the project file. This will allow filenames and paths specified relative to the project file path in step and macro properties to work correctly when building. If unchecked, the current directory of the process will not be modified. The `PROJDIR` system macro can also be used to specify paths relative to project path.

*Note:* The Set Current Dir action can also be used to set the working directory.

**Persist build status when saving project files:** Normally, the build status of a project is not saved in the project file. Check this option to cause changes in the build status to mark a document as modified and for the build status to be saved with the project when saved from the GUI app. This can be useful for continuing an automated build from a point of failure. Be aware that the project will not automatically be saved in this case (the GUI App will prompt that changes have been made, and the console app will discard changes without prompting) -- it is recommended to add Run Script steps to the project which saves the project with status on failure or without status on success if this option is enabled (also see the `SaveStatus.bld` sample).

**Include environment variables in system macros:** If checked (the default), the operating system environment variables are included in the Visual Build system macros. If you don't need these variables and want to shorten the list of system macros, uncheck this item. A few environment variables (`COMSPEC`, `TEMP`, and `WINDIR`) will still be defined as system macros even if this option is unchecked).

**Case sensitive comparisons in conditional build rules:** If checked, comparisons in build rules are performed with case sensitivity (the default is non-case-sensitive).

**Re-evaluate all conditional build rules for each step:** If checked, all build rules that apply to the current step (its rule and the rule of any parent steps if nesting of build rules is enabled) are re-evaluated before each step is built; if not checked, only the current step's rule is evaluated, and the previous evaluation result of any parent steps' rules are used.

**Build failure steps when build is stopped:** If checked, failure steps will be built if the build is aborted (and the step is configured to build failure steps). If the build is aborted again while building failure steps, the build will stop immediately without completing all failure steps. If this option is unchecked, failure steps will only be built if an error occurs while building a step.

**Fail overall build if any steps failed during the build:** If checked, the overall build completion status will be set to failed if any steps failed (but the build continued) during the build.

**Delete temporary macros after successful build:** If checked, any temporary macros will be deleted when a project successfully completes building. This is useful if you are prompting for temporary macro values; this will cause the value to be prompted for again if the project is built again.

**Delete temporary macros on rebuild or reset:** If checked, any temporary macros will be deleted before a build is reset or a rebuild is performed.

### 2.7.3.2 Logging

These options from the application options dialog configure global file log settings for builds:

**Enable file logging:** When checked, build output is written to the specified log file in addition to the Output pane. Use the Browse button to choose a file, View to view the current log file in viewer (configured in the user options), Launch to open in its associated application, or Delete to delete the log file if it exists.

**Log Filename:** The name of the log file to write build output to if file logging is enabled (this value is available in the system macro LOGFILE if logging is enabled). This is a global setting that affects all builds, unless the LOGFILE macro is overridden by a project. Project-specific log files can also be used for every project automatically by entering %PROJDIR%\%PROJROOT%.<ext> in this field. These system macros will be expanded when a project is built and the path and root name of the project file will be used for the log file.

The log filename can also be overridden for a given project in the Project Properties dialog or by passing in a temporary LOGFILE on the command-line. This can be useful for creating *project-specific log files* or to dynamically modify the target filename (for instance, to temporarily disable file logging). See the Logging.bld sample for more details.

**Format:** Determines the format that will be used to generate the log file. The available options are *Text* and *XML*. *Text* log files are similar to the output displayed in the Output pane (including a timestamp for each step), and additional logging options are available for this format.

*XML* log files are written in a hierarchical XML format, with elements for each build and step. The log file is appended to as steps are built, so until the build completes, the closing step, build, and log tags (elements) will not yet exist. The Transform XML Log action can be used to add the closing tags (making the log a valid XML document that can be sent within a build), to filter the output, and to convert XML logs into HTML documents. The Logging.bld sample demonstrates several techniques that are useful when using the XML log format (including deleting an XML log file before a build and disabling/enabling file logging during a build via code in script events). For easier browsing of the file in Internet Explorer, XML log filenames can be given a .xml extension.

*Note:* To change the log file format to XML before file logging starts, use this VBScript code in the vbld\_BuildStarting script event:

```
Application.Options.LogFormat = "XML"
```

**Log all failure step output:** If checked, even if a message is logged from a failure step with a higher level than the above Log level setting, the message will still be written to the log. If unchecked, such messages logged from failure steps will not be written to the log.

**Delete log file at the start of each build:** If checked, the log file will be deleted at the beginning of each build. Otherwise, the log file is appended to with each build. When using XML logging and master/child projects that use the same log file, this option should not be checked, because it would cause the log file to be deleted at the beginning of the master and each child project.

**Log default property of each step:** If checked, each step's default property will be logged when it is built. This can be useful for debugging purposes.

**Log build rule when a step is skipped or repeated:** If checked, when a step's repeating build rule is processed or when a step with a build rule is skipped, the details of the build rule will be logged.

*Notes:*

- Logging can be disabled for individual steps via the *Do not log action output* step property.

- It is recommended that any active scanning anti-virus software be disabled on the build box, as this can interfere with Visual Build writing to its log file (and also slows down builds).
- If the log file cannot be written to, the error information will be written to the Windows **Event Viewer** Application tab (available in Windows Administrative Tools).
- To make the build log accessible to other users, simply map an IIS or web server virtual directory to the path containing the log files, and the log files can be accessed via any web browser.

### 2.7.3.3 Logging (More)

These options from the application options dialog configure additional global log settings for builds:

#### Text File and Console App Logging Options

These options affect what is written to the build log file when using the Text logging format and the console when building with the console app. Options for XML format are configured on the Filter tab of the Transform XML Log action.

**Log level:** Determines which log messages will be logged to the log file when building a project (messages with a level the same or lower than the level specified here will be logged). This can be useful to reduce the amount of build activity logged during a normal build. And if a step fails during the build, the FAILSTEP\_OUTPUT macro (available from Failure Steps) will contain all logged messages for that step, even if they were not logged due to the Log level setting.

**Logging of step starting events:** Determines the logging behavior of step starting build events to a text file.

**Include step numbers:** If checked, each step's number is also logged (if logging step starting events).

**Log skipped steps:** If checked, steps skipped during a build will be logged to the file.

#### Other Logging Options

**Strip carriage return/linefeed characters within log messages:** If checked, new line characters are removed within each log message so that each message stays on the same line in the file. If unchecked, the output contents are written to the log without changes.

**Convert double quote characters to single quotes in step output macros:** Whether to convert double quotes in the LASTSTEP\_OUTPUT and FAILSTEP\_OUTPUT system macros to single quotes. Checking this option this can be useful if the macro will be referenced in script expressions, but it can cause problems for quoted values in XML output containing single quotes.

**Escape special characters in step output macros:** Whether to escape characters in the LASTSTEP\_OUTPUT and FAILSTEP\_OUTPUT system macros. Bracket characters [ and ] normally denote a script expression to be inserted into a field, and the percent sign character % is normally used around a macro name for its value to be expanded within the field. If this option is checked, these special characters in step output will be doubled [ [ ] ] %%.

*Note:* This option should be checked if the output macros will be referenced in step fields (i.e., %LASTSTEP\_OUTPUT%) so that any special characters will be treated as literal characters.

**Echo output from console programs called from Console app:** Determines whether to echo the output of console programs called from a build using the Console application. If checked, the output of any console programs called by a Run Program or VisBuildPro Project step or any action that calls the Builder object's RunProgram or RunProgramEx methods will be echoed to the console that VisBuildCmd is running in. If unchecked, any output of such chained programs will not be echoed.

#### 2.7.3.4 File Locations

The File Locations dialog is accessed from the Tools menu. It displays the current locations for the Visual Build global application settings

**View:** Opens the selected file in the configured viewer.

**Launch:** Opens the selected file in its associated application or opens the selected folder in Explorer.

**Reload:** If clicked, Visual Build recreates or updates the predefined global macros. This can be useful to restore the default values, since they can be modified by the user and because Visual Build attempts to initialize many of these variables according to the installed location of third party tools (i.e., DOTNET\_DIR, DOTNETSDK\_DIR, etc.). This feature can be used to reinitialize the values after installing one of these tools or to restore the original values.

#### 2.7.3.5 Advanced

This tab of the application options dialog configures the following options:

**Enable logging of severe errors to Windows Event Log:** Determine whether severe errors (for instance, errors that occur writing to the build log or in silent mode) are written to the Windows Event Log.

**Always show GUI applications launched from build:** When Windows GUI applications are called from a build (via the Run Program or derived actions, the RunProgramEx method, etc.) and this option is checked, they will be displayed (not hidden) even if the step is configured to hide the application window. If unchecked, the hidden state will honor what was specified by the step or action.

**Maximum step output macro length:** Specifies the maximum length of step output to store in the LASTSTEP\_OUTPUT and FAILSTEP\_OUTPUT system macros. Any step's output exceeding this length will not be appended to the output macros. A value of 0 will add all step output, regardless of total length.

**Write byte order mark (BOM) when saving files:** Whether to include byte order mark on XML files that are created or updated.

**Use UTF-8 code page for console app:** By default, the console application and RunProgram / RunProgramEx and derived actions use the current code page for processing of program input and output. If this option is set to *True*, UTF-8 conversions will be performed instead. Enabling this option can improve capture and logging of Unicode text from programs that are called from a build, but it requires Windows Vista or later (console program output may not be captured properly for older versions of Windows). When running the console application from a Command Prompt, a Unicode font must also be used for proper display of Unicode text.

**Encrypt password property values:** Determines whether password property values will be stored encrypted in the project file and global macros file when saving.

**Obscure password property values:** Determines whether the value of encrypted step and macro properties will be hidden in the step and macro panes and build output and log file.

**Type libraries to load for script:** Specifies ProgIDs or GUIDs of type libraries to load for scripting (one per line; append ;major;minor to specify a library version). This allows using constants from type libraries such as *Scripting.FileSystemObject* (i.e., ForReading, ForWriting, etc.) within script code without defining them manually.



## 2.7.4 Change Password

The Change Password dialog is accessed by choosing the *Change Password* item on the File menu. It is used to set, change, or remove a user-defined key for encrypting protected data in the project file.

If the related option is enabled and no (blank) project password is specified, password step properties (properties in actions marked as containing a password) and encrypted macro values will be stored encrypted (using 256-bit AES encryption and a static initialization vector [or a unique initialization vector if the UniqueEncryptionIV application option is True]) by Visual Build with a custom private key when saved. If a project password is specified in this dialog, encrypted values in the project file will also be encrypted with the user-provided key, and the key must be provided in order to open the project file.

Encrypted values are also obscured (displayed with asterisks if the related option is enabled) when displayed in the Step Properties dialog, Macro Properties dialog, Insert Macro dialog, Step Panes, or Macros Pane, when logged, and no tool tips are displayed for encrypted values.

## 2.7.5 Change Edit Password

The Change Edit Password dialog is accessed by choosing the *Change Edit Password* item on the File menu. It is used to set, change, or remove a password required to modify the project file.

After opening a project file with an edit password in the GUI App, the project can only be built and cannot be modified unless the edit password has been provided. When specifying an edit password, it is best to save the file in compressed format (with .bldx extension) to prevent removal of the edit password from the project file.

*Note:* The project can still be modified via the object model even if an edit password has been specified.

## 2.7.6 Customize

The Customize dialog is used to customize toolbars, menus, keyboard shortcuts, and the visual appearance of Visual Build. This dialog can be displayed via *Tools | Customize* on the menu bar or by right-clicking a toolbar or the menu bar and choosing Customize.

- Toolbars
- Commands
- Menus
- Keyboard
- Options

*Note:* Customizations are stored in the registry under the key `HKEY_CURRENT_USER\Software\Kinook Software\Visual Build Professional 9\Layouts`. To reset these customizations to their defaults, exit Visual Build, download this file and run `ResetLayouts.reg` by double-clicking (accepting all prompts).

### 2.7.6.1 Toolbars

The Toolbars tab of the Customize dialog is used for creating, deleting, renaming, showing, and hiding toolbars and menus in Visual Build. To restore the defaults for a toolbar or menu bar, select the menu or toolbar in the list and click Reset.

*Note:* The menu bar cannot be hidden.

### 2.7.6.2 Commands

The Toolbars tab of the Customize dialog can be used to modify the Visual Build menus and toolbars.

- To remove an existing menu item or toolbar button, click and drag the item or button and drop it outside of the menu or toolbar.
- To add a new button or menu item:
  1. Select the category of the command you want to add (use All Commands if unsure).
  2. Drag and drop the command onto the target toolbar or menu.

*Note:* Menus and toolbars can also be customized without displaying the Customize dialog by holding the Alt key while dragging menu items and toolbar buttons to move them to another location.

### 2.7.6.3 Menus

The Menus tab of the Customize dialog can be used to modify the Visual Build context (right-click) menus.

- Select a context menu to customize.
- To reset the menu to the defaults, click Reset.
- To remove a menu item, drag and drop the menu item out of the menu.
- To add a menu item:
  1. Switch to the Commands tab.
  2. Select the category of the command you want to add (use All Commands if unsure).
  3. Drag and drop the command onto the context menu.

### 2.7.6.4 Keyboard

The Keyboard tab of the Customize dialog is used to view and customize the keyboard shortcuts associated with commands. To view the key assignment(s) for a command, first choose the category and command. To assign a new shortcut, type the shortcut key into the *Press new shortcut* field and click Assign. To remove an assignment, select its assignment and click Remove. Click Reset All to restore all keyboard shortcuts to the default values.

*Note:* The currently assigned keyboard shortcut for each command is also displayed on the menu bar and in the command's toolbar tool tip if the related option is enabled.

## Fixed Shortcuts

Most keyboard shortcuts can be configured as described above. The following shortcuts are not configurable by the user:

Ctrl+C: Within dialogs (step properties, macro properties, etc.) and when editing grid values, copies selected text value.

Ctrl+M: Copies the current field's expanded value (with all macros expanded) to the clipboard.

Ctrl+Shift+M: Copies with macros expanded and with script evaluated.

Ctrl+Enter: Within a step or macro properties dialog multi-line edit control, if Enter inserts a new line in the control, Ctrl+Enter will accept changes and dismiss the dialog; if Enter accepts the dialog,

Ctrl+Enter will insert a new line into the edit control.

## Special treatment of the Shift and Control keys

### Shift

- Causes script expressions within a step property to be evaluated and displayed in the tool tip.
- Toggles the checked status of only the clicked-on step in the step grid (by default, child steps are also toggled).
- Forces items to be moved with drag and drop.
- Multi-selects all items between last selected and clicked item in multi-selection grids.

### Control

- Causes all child steps to be expanded when expanding collapsed steps.
- Causes arrow keys to change focused item in grid without changing selection.
- Forces items to be copied with drag and drop.

### Script Editor

*Note:* There are two ways to customize keyboard shortcuts in the Script Editor dialog: 1) Using the editor's Customize dialog (*Tools | Customize* on the dialog's menu) as described above, or 2) via the Keyboard tab of the Script Editor options dialog (*Tools | Options* on the Script Editor menu). The shortcuts configured via the Customize dialog take precedence, but not all editor commands are configurable from this dialog. To view or customize additional editor shortcuts, use the Options dialog. To view the key assignment(s) for a command in the Options dialog, select the command (or type a shortcut into the *New key assignment* field to view its currently assigned command). To assign a new shortcut, type the shortcut key into the *New key assignment* field and click Assign. To remove an assignment, select its assignment and click Remove. Click Reset All to restore all keyboard shortcuts to their default values.

#### 2.7.6.5 Options

The Options tab of the Customize dialog configures global options related to toolbars and the menu bar:

**Always show full menus:** Checking this box will ensure all menu items are always displayed when a menu is selected. If not checked, the application will initially only show the menu items that have been recently viewed.

**Show full menus after a short delay:** If this box is checked (and enabled), full menus will be displayed after a short delay. If this box isn't checked, the only way to see the full menu is to click the bottom Expand button displayed with each personalized menu (this button disappears when the full menu is displayed).

**Paint theme:** This combo box allows you to select one of several available themes that the visual appearance of the application.

**Large icons:** Checking this box will show larger than normal toolbar and Menu Bar icons.

**Show ScreenTips on toolbars:** When this box is checked, tool tips will be displayed when the mouse is hovered over toolbar buttons.

**Show shortcut keys in ScreenTips:** When this box is checked (and enabled), button tool tips will include the associated keyboard shortcuts for commands that have a shortcut.

**Menu animations:** Specifies the animation effect to use when displaying menus.

#### 2.7.7 Edit Password

The Edit Password dialog is displayed when attempting to modify a project that has an edit password assigned. The correct password must be provided in order to modify the project.

#### 2.7.8 Find/Replace

The Find and Replace dialogs are accessed from the Edit menu or via several shortcut keys.

The Find dialog can be used to search for text within steps, macros, and script. The Replace dialog can be used to find and replace text within steps and macros (excluding system macros). The Script Editor and Run Script action provide their own Find and Replace dialogs, which support regular

expressions and search and replace within script code.

### 2.7.9 Go To Step

Use this dialog to quickly jump to a particular step in the current step pane. The dialog is accessed from the Go menu or via a shortcut key. User options configure the behavior for displaying steps in the dialog.

**Step:** The name, action, or index of the step to go to (required). If a partial name is entered, the list will be filtered to show steps that match what has been typed. The up and down arrow keys can be used to navigate in the list while focus is in the Step field, and pressing Enter will go to the selected step.

**Step list:** A list of steps in the current step pane, filtered by what is typed in the Step field. To choose the selected step, double-click on it or press Enter. The step will be selected in the step pane.

### 2.7.10 Insert Macro

Displays a list of all defined macros for quickly inserting a reference to a macro into the current field.

**Filter:** The Filter field can be used to quickly filter the list to locate a specific macro. As text is entered, the list is filtered to match only those macros matching what has been typed.

The up and down arrow keys can be used to navigate in the list while focus is in the Filter field, and Enter can be used to insert a reference to the selected macro.

**Macro grid:** A categorized grid of available macros. Type a name or click with the mouse to select a macro. To insert a reference to the selected macro, double-click on it or press Enter. The macro reference will be inserted into the last focused field at the cursor position.

When the mouse cursor is held over a macro, its description will be shown in a tool tip. The dialog can be resized, additional columns can be added by right-clicking the column header, and columns can be reordered and resized by dragging the column or column border.

**Clear:** Clears the Filter field and repopulates the action list with all registered actions.

**Properties:** Displays the Macro properties dialog for editing the selected macro.

### 2.7.11 Macro Properties

The Macro Properties dialog can be used to add new macros and modify existing macros. To add a new macro, switch to the Macros pane, and click the Insert icon on the toolbar, choose *Edit | Insert* on the menu bar, or press Insert (if the related option to edit properties via the properties pane when inserting is unchecked). To edit an existing macro, click the properties icon on the toolbar, select *View | Properties Dialog* on the menu, or press Enter.

Macros can be used in most step fields and within other macros. You can define your own macros, use existing environment variables, or use the predefined Visual Build macros in your projects. Macros replace a particular string in the project with another string. Using macros, you can:

- Make your project generic
- Define common commands for use in projects
- Use predefined commands that are available as part of Visual Build
- Create a generic project that can be used with variable parameters
- Specify options for commands used in multiple project steps

### Dialog Fields

**Name:** Required. Used to reference the macro in the project steps or other macros (i.e., %MACROX% in a step field will be expanded to the value of the macro named MACROX when the step is built). Macros can contain any characters other than the left/right parenthesis, comma, percent sign (%), or space. Macro names are matched without case sensitivity.

**Parameters:** Specifies parameters that can be passed to the macro. Parameters are separated by commas (optional).

*Note:* This field is disabled by default, but it can be enabled by unchecking the related option.

**Value:** The value to replace a reference to this macro with. If the macro has parameters, each parameter must be used at least once in the macro value. Macros can reference other macros to 20 levels of nesting (although a macro cannot reference itself). To reference a macro in a macro value or step field, place a percent sign (%) sign around the macro name (i.e., %DEVSTUDIODIR%). If other macros are referenced, a tool tip shows the expanded macro value when the mouse cursor is held over the field.

*Note:* Bracket characters [ and ] and the percent sign character % within a macro's value have special meaning in a macro value.

**Description:** A comment describing the purpose of the macro (optional).

**Category:** A category to group the macro in when displayed in the macros pane (optional).

**Add to environment variables:** If this box is checked, the macro's value will be added to the environment variable for any external programs that are called in subsequent steps.

**Encrypt macro value:** Determines whether the macro value is encrypted when displayed and when stored in the project file. Unchecking this option for an existing encrypted macro is disabled.

## Buttons on the Dialog

**Insert Macro:** Displays the Insert Macro dialog to add other macros to the current macro's value. A list of all defined macros is displayed, and double-clicking the macro inserts it into the field at the current cursor position. Macro names can also be entered manually. A macro is indicated by placing a percent sign (%) on each side of the macro name (i.e., %DOSCMD%).

**Browse:** Displays a file selection dialog for inserting a filename into a macro value.

**Shell:** Displays a popup menu for performing an operation on the currently selected filename or folder. This provides a handy way to open a file in its associated application (Launch), view a file in the user-configured viewer (View), or to open a folder in Windows Explorer (Explore).

### 2.7.12 Password

The Password dialog is displayed when opening a project containing properties that have been encrypted with a user-provided key. The encryption key must be entered in order to decrypt any encrypted step properties or macro values in the project. The password can also be provided via the command-line or object model.

### 2.7.13 Print/Print Preview

Print and Print Preview is provided for previewing and printing Visual Build projects (steps, macros, output, etc.). These functions are accessed from the File menu. By default, the active pane is printed or previewed. The type of output and other settings can be configured via several print options. Script code can also be printed from the Script Editor.

In the Page Setup dialog (accessed by clicking the 5th button from the right on the preview dialog), the values in % characters (i.e., *%longdate%*) are variables that display a computed value when printed. The following table shows the available variables:

<i>%page%</i>	Current page number
<i>%total%</i>	Total page count
<i>%time%</i>	Current time (displayed using the short form without seconds according to the user's current regional language settings)
<i>%date%</i>	Current date displayed using the short date format according to the user's current regional language settings
<i>%longdate%</i>	Current date displayed using the long date format according to the user's current regional language settings

## 2.7.14 Project Properties

The Project Properties dialog is used to view or edit comments stored with a Visual Build project, display statistics about the project file, and to configure project-level file logging. It is accessed from *File | Properties* on the menu bar.

- Comments
- Properties
- Logging

### 2.7.14.1 Comments

Used to view the project filename and view or edit the comments for a project. This can be useful for providing instructions for a build when other users build the project. If the option is enabled in the User options dialog, it is also displayed automatically when the project is opened (unless it is being opened for an automated build).

### 2.7.14.2 Statistics

This tab displays the project path+filename, size, and other statistics about project, global, and system steps, macros, and script code. A button to display the Explorer properties dialog for the file is also provided, which can be used to change the read-only flag or view other file properties.

### 2.7.14.3 Logging

This tab configures project-level file log settings for a build.

**Project-level file logging:** When checked, all build output is also written to the specified log file. Use the Browse button to choose a file, View to view the current log file in the configured viewer, and Delete to delete the log file if it exists. If enabled, these settings will override the global settings for the current project; otherwise the global settings will be used.

**Log Filename:** The name of the log file to write build output to if file logging is enabled (this value is stored in the project macro LOGFILE if logging enabled). This field can contain macros or script expressions.

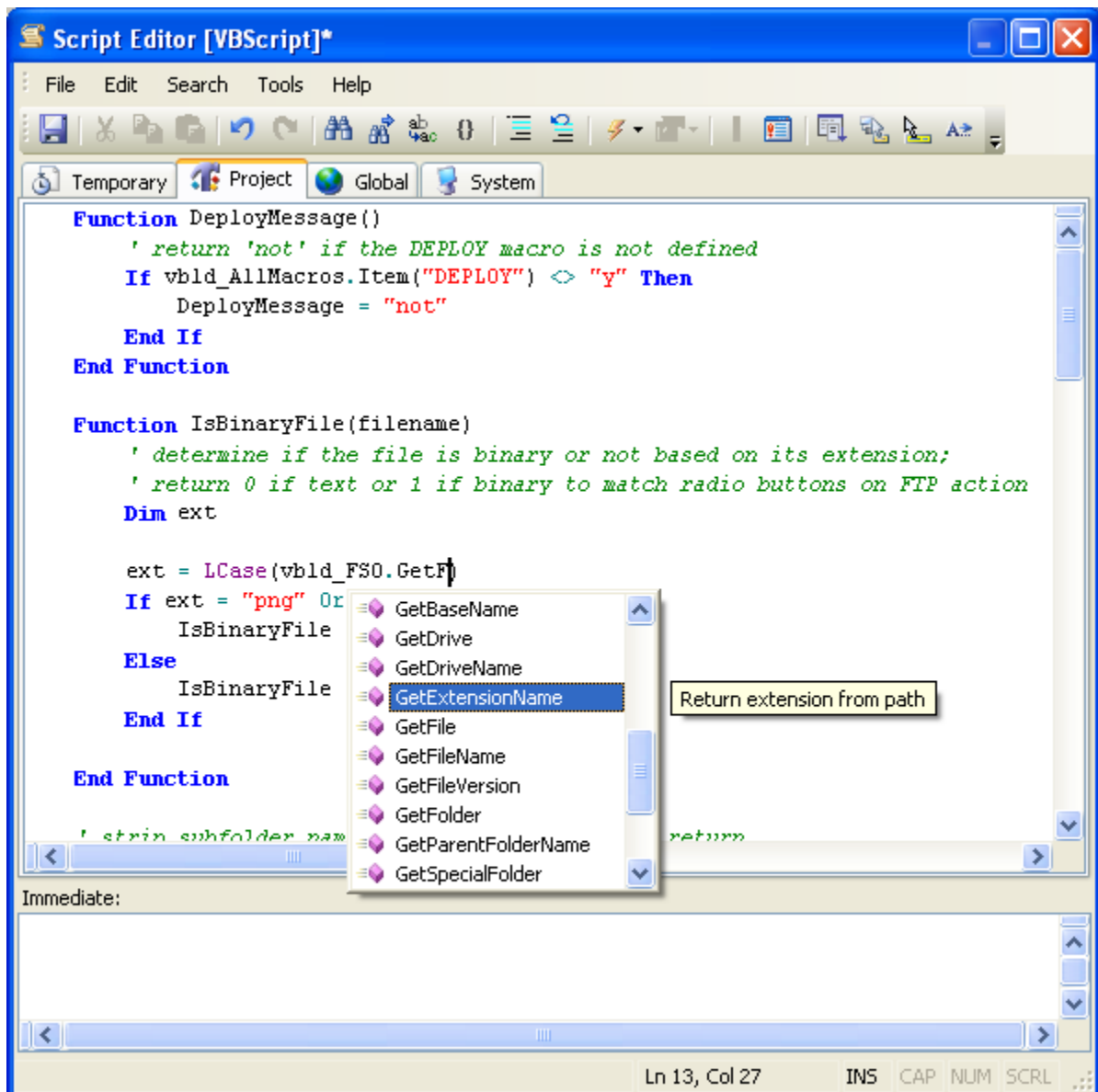
*Notes:*

- These settings can also be overridden by a temporary LOGFILE macro passed on the command-line when the project is built.
- Other file logging options, including the file format (text or XML), are configured in Application options.
- Enter an empty string for the log filename to disable file logging for the current project even if it is enabled globally.

## 2.7.15 Script Editor

The Script Editor allows viewing and editing of scripts, step events, and user action script code within Visual Build. It can be accessed via the View menu, and from the Step properties dialog. The Script Editor has its own menu and toolbar, which can be customized as desired. Keyboard shortcuts are also configurable.

Script functions defined in the Script Editor can be used within script expressions in step fields, in the Run Script action, and any event functions called during a build.



There are five (5) types of scripts, with one tab entry for each type:

**Temporary:** Scripts that exist only for the current Visual Build instance. They are created by adding them to the Temporary tab or by loading them from a file during a build using the object model.

**Project:** Saved with the project file.

**Global:** Global across all projects; stored in the `VisBuildPro.Global.scripts` file in the configuration files path.

**System:** Built-in scripts that are installed with Visual Build, stored in the `VisBuildPro.System.scripts` file in the install path, and are overwritten during installation.

**Step/Action:** Used to define event functions for an individual step or to define the script code for a user-defined action. This tab is only displayed when the Script Editor is launched from within the step properties dialog or the component tab of the action properties dialog.

Unlike macros, where macros of the same name can override another based on their precedence, all the script of each type is concatenated together and added as global items that are accessible to all script code. Script functions names prefixed with `vbld_` are reserved for use by the Visual Build system scripts.

*Notes:*

- To create a temporary macro in script code (VBScript), use

```
Set objMacro = vbld_TempMacros.Add("MACRO_NAME", "macro value")
or
Set objMacro = Application.Macros(vbldMacroTemporary).Add("MACRO_NAME",
"macro value")
```

- The `Script.bld` sample demonstrates common script functionality, including creating or incrementing a macro value, if/else logic, script expressions in build rules, etc.
- Script code entered in the script editor cannot reference macros directly -- use the script code `Application.ExpandMacrosAndScript("%MYMACRO%")` instead of `"%MYMACRO%"`. However, macro references entered in the Immediate window will be expanded, and script entered in the Run Script action and within brackets in other step properties can reference macros directly.
- Script code entered on the Step tab is only in scope for step script event methods.

## Immediate Window

The Immediate window is useful as an aid in debugging script code and script expressions. Type an expression in the field, optionally prefixing with `?`, and press Enter or click the Evaluate button to evaluate the script code (the result is displayed in the Immediate window below the typed expression). Use the Insert item on the File menu to insert the script from the current line in the Immediate window into the current control in the Step properties dialog (if the Script Editor was accessed from the Step properties dialog) or to insert into the Watches pane. Choose *Search | Go To Last Error* on the menu bar to navigate to the script code line where the last error occurred.

*Notes:* This dialog requires that the [MSXML parser](#) (version 3.0 or later) be installed for syntax highlighting and code completion functionality.

### 2.7.16 Step Properties

The Step Properties dialog can be used to add new steps to a project or modify existing steps. For new steps, it is accessed by double-clicking an action on the Actions pane or by choosing Insert on the Edit menu (if the related option to edit via the properties pane when inserting is unchecked). For an existing step, it is accessed from the Properties button on the toolbar or the Properties Dialog item on the View menu. Multiple steps can be selected and edited simultaneously. If all steps have the same action, all the property tabs will be displayed, otherwise only the General tab will be available. All properties that are modified will be updated in all selected steps (the initial dialog will show the properties of the first selected step). When the mouse cursor is held over a property, its expanded value will be shown in a tool tip.



The step action is displayed in the dialog caption. Each step contains General and General (More) tabs for standard properties and may contain additional tabs specific to the action for that step.

## General Tab

## More Tab

## Buttons on the Dialog

*Note:* Bracket characters [ and ] and the percent sign character % within step fields have special meaning in a step field.

### 2.7.16.1 General Tab

The General tab of the Step Properties dialog is used to configure common step properties.

**Name:** Used to identify the step when building (required).

## Build Conditions

This section of the step properties dialog General tab is used to define the conditions for whether a step will be built or skipped during a build. This is a powerful capability that allows selective execution of project steps based on the value of a build profile, macros, script, or environment variables, implementing of loop constructs (performing a step or set of steps a certain number of times, until some condition is reached, once for each line in a file, etc.), and more.

**Include in build:** If checked, the step and its child steps will be considered for building, depending on the following settings. If unchecked, the step will not be included in the build.

**Profile:** Specifies the step's profile (optional). Only steps with an empty profile or a profile value matching the build profile will be included in the build. The step profile can be a literal string or a regular expression (for instance, *Debug.\** to match all build profiles beginning with *Debug*, *.\** to match all, or *Staging|Production* to match either the *Staging* or *Production* or build profile). A single asterisk (\*) can also be used as a wildcard to match any profile value.

If the step is included and matches the build profile, a conditional build rule can also be defined to determine whether the step will be built.

If *Build this step and children when checked and profile matches* is selected, no build rule will be evaluated. Select *Build only if macro or expression* to define a build rule for the step. Enter the expression to evaluate, the comparison to use, and the value to compare with. When building, Visual Build evaluates the rule, and builds the step and its children if the rule evaluates to a true condition and skips the step if it doesn't. The following rule comparisons are supported:

<u>Comparison</u>	<u>Behavior</u>
-------------------	-----------------

is undefined	Builds the step if the macro (i.e., %MYMACRO%) does not exist
is defined	Builds the step if the macro is exists
contains	Build if the expression contains the string in the following field (macros and/or script can be used in either field)
is equal to	Build if the expression is equal to the string in the following field
is not equal to	Build if the expression is not equal to the string in the following field
does not contain	Build if the expression does not contain the string in the following field
is true	Build if the expression is true (a non-zero number or the string <i>True</i> ) -- useful when evaluating script expressions
is false	Build if the expression is false (zero or the string <i>False</i> )

Settings in the Application options determine whether comparisons are case sensitive.

*Note:* When entering a macro in the *macro or expression* field, the macro name must be surrounded by percent signs (%MYMACRO%) so that the macro's value is used in the comparison. Otherwise, a literal comparison of the string that was entered and the comparison value will be performed.

For more complex rules, a script expression can be used determine whether the step should be built. Any valid script expression can be used, and the expression can call functions defined in the Script Editor. Script expressions are marked by bracket characters: [script code here].

## Repeating Rules

A while **loop** can be implemented by checking *Repeat step while condition is true*. If the rule evaluates true, the step and all child steps will be built; the rule will then be re-evaluated and the steps repeated until the rule evaluates false (a macro can be modified in a child step to increment a counter or set the condition to false as appropriate, or a script function could be called which determines whether to continue the loop). A repeating set of steps can also exited with the Exit action or by the step failure exit build/subroutine option.

*Note:* A loop can also be implemented with the Loop action.

## Additional Details

Conditional building normally applies to the current step and all child steps (steps below and indented from the step), so a build rule can easily be applied to a block of steps. Normally, rules are nested (if a child step also defines a build rule, the parent and child step's rules must evaluate true).

*Note:* The *Include in build* setting, build profiles, and conditional build rules are not evaluated when performing a Rebuild Selected or when clicking the Test button on the step properties dialog; see the Methods of Building help topic for other ways to test iterating steps or steps with build rules.

**Pause before and after building step:** Specifies a delay (in seconds or fractions of a second [i.e., 0.25]) to pause before and/or after building the step (optional).

### 2.7.16.2 More Tab

The More tab of the Step Properties dialog is used to configure additional common step properties.

**Description:** A description of the step (optional). It will also be displayed in a tool tip in the Step pane when the mouse is over the Step Name column.

**Disable logging of action output:** If checked, any logging the step's action performs (to a file or the Output pane) will be ignored and not logged.

*Notes:*

- Error level messages will still be logged even if this option is checked.

- Logging from any of the step's script events will not be suppressed.
- The LASTSTEP\_OUTPUT macro will contain any output logged by the action even when this option is checked.

## On Step Failure

This section of the step properties dialog can be used to configure the action(s) to perform if the step fails to build. It provides flexibility for handling error conditions within a build.

**Retry x times, pausing y seconds:** Specifies how many times to retry building the step if it fails to build and a pause between retries (optional).

*Note:* While retrying, the current retry number will be stored in the FAILSTEP\_RETRY temporary macro.

**Build failure steps:** If this item is checked, the specified Failure steps subroutine (or all failure steps if blank) will be built if this step fails to build.

**Stop / Continue / Exit subroutine/build:** Determines the action to take after building any failure steps (if specified above). *Stop* causes the build to stop (and exit if building with the Console app or the GUI app if launched from the command-line with the silent flag); *Continue* causes the build to continue processing any remaining steps; and *Exit* returns from a subroutine or aborts the build if not in a subroutine.

**Terminate action if not completed after x seconds:** If a non-zero value is specified and the action has not completed within that number of seconds, the build component will attempt to terminate the action. The build status of the step will be *Terminated*.

*Note:* Termination of an action will succeed only if the action periodically checks or waits on the CancelEvent property (this is automatic for the built-in actions and user actions that call RunProgramEx).

### 2.7.16.3 Dialog Buttons

**OK:** Accepts all changes and closes the step properties dialog.

**Cancel:** Discards any changes since Apply was clicked and dismisses the dialog.

**Apply:** Commits any changes to the underlying step object.

**Test:** Accepts any changes, closes the dialog, and rebuilds the current step.

*Note:* The Test button selects the current step and performs a Rebuild Selected, which ignores conditional build rules; see the Methods of Building help topic for other ways to test step build rules.

**Insert Macro:** Displays the Insert Macro dialog to add macros to a field. A list of all defined macros is displayed and double-clicking the macro inserts it into the field at the current cursor position. Macro names can also be entered manually. A macro is indicated by placing a percent sign (%) on each side of the macro name (i.e., %DOSCMD%). If the macro takes parameters, enter them in parentheses after the macro, separating the values with commas (i.e., %ATTRIB(-r, \*.bat)%).

*Notes:*

- A literal percent sign character (%) can be entered into a field by entering two percent signs (%%); within parentheses, a literal parenthesis or comma character can be entered by typing two parenthesis or comma characters. When the mouse cursor is held over one of these fields, all

macro values are expanded and shown in a tool tip. This is the actual string that will be used when building the step.

- To copy the current field's expanded value (with all macros expanded) to the clipboard, type Ctrl+M. Ctrl+C copies the value without expansion. Ctrl+Shift+M copies with macros and expanded and with script evaluated.

**Browse:** This button displays a dialog to browse for a file or folder to insert into the current field. If the file or folder is in the same folder or a sub-folder of the project file, the project file path will automatically be replaced by the %PROJDIR% system macro. Using this macro allows projects to be defined generically so that they will build correctly regardless of the path that the project file is placed in.

**Script Editor:** Displays the Script Editor window for modifying step event script code and other scripts, evaluating script expressions, and inserting script expressions into a field. If any script events are defined for this step, a plus sign (+) will be added to the Script Editor button caption.

Script code within a field is indicated by placing brackets around it (i.e., [ 5+5 ]); the script language used is the default script language set in the application options dialog. When using the Insert button on the Script Editor dialog, the brackets are added automatically. Normally, script code is not evaluated for display in the field's tool tip (this can be useful to see the script code after any macros have been expanded). Holding down the Shift key before moving the mouse cursor over the field will cause the script code to be evaluated and displayed in the tool tip.

*Note:* To enter a literal bracket character, type two bracket characters [[ or ]].

**Previous/Next Step:** Used to navigate between steps in a project without closing the Step Properties dialog. If changes have been made to the current step, the user will be asked if the changes should be kept or discarded.

**Shell:** Displays a popup menu for performing an operation on a selected filename or folder in the current field. This provides a handy way to open a file in its associated application (Launch), view a file in the user-configured viewer (View), or to open a folder in Windows Explorer (Explore). Any macros or script are expanded to determine the filename to be used. If there is no selection in the field, the entire field value will be used.

## 2.7.17 User Options

This dialog is used to configure user-specific settings for the Visual Build GUI application, and is accessed from *Tools | User Options* on the menu bar.

- General
- Projects/Steps
- Display
- Columns
- Build
- Output
- Colors
- Print/Backups
- Miscellaneous

*Note:* These options are stored in the registry under the key `HKEY_CURRENT_USER\Software\Kinook Software\Visual Build Professional 9\Options`. To reset all user options and other user customizations to their defaults, exit Visual Build, download this file and run `ResetAll.reg` by double-clicking (accepting all prompts).

### 2.7.17.1 General

The General options tab of the user options dialog provides the following options:

**Reload last project at startup:** Determines whether the previously opened project is opened the next time Visual Build is started (ignored if a Visual Build project file is opened from Explorer or a filename is specified on the command-line). Even if checked, reloading the last opened project can also be disabled by holding down Shift during startup.

**Remember selected steps when reopening projects:** If checked, the last selected steps will be reselected then next time a project is opened.

**Show project properties when opening project:** Determines whether the project comments dialog is displayed when a project is opened (only if comments have been entered for that project). Even if checked, the dialog is not displayed when Visual Build is called from the command-line to build a project.

**Display full path of project file in title bar:** If checked, displays full path and filename of the current document in the application title bar; if unchecked, displays the filename without path.

**Reselect last selected action in Actions pane on startup:** If checked, the last selected action in the Actions pane will be reselected when Visual Build starts.

**Select text when focusing Actions pane edit control:** If checked, the text in the Actions pane edit control will always be selected when focused.

**Show application icon in system tray:** Determines whether the Visual Build GUI adds a system tray icon when running. The tray icon will indicate the current build status while building. Double-clicking the system tray icon will activate that Visual Build instance (and unhide if minimized to the tray). Right-clicking the system tray icon will display a menu of commands related to that Visual Build instance.

**Minimize to system tray:** If checked and the previous option is also checked, when the Visual Build main window is minimized, its Start menu entry will be removed.

**Check for updates every x months:** Specifies how often to check for application updates (enter 0 to disable update checking). If enabled, the GUI app will check for updates at startup every # months specified (launching the update check in the default browser).

### 2.7.17.2 Projects/Steps

The Projects/Steps tab of the user options dialog is used to configured settings that apply to projects and Step panes.

**Use Properties window when inserting steps and macros:** If checked the modeless properties pane will be used for editing properties of new steps and macros. If unchecked, the modal Properties dialog will be used.

*Note:* Some properties can only be edited in the step properties dialog, and the dialog will be opened to edit these properties.

**Create root step for new projects:** If checked, creates a root Group action step when creating a new project. It can be useful to create a root step that all other steps are children of (for instance, to quickly check or uncheck all steps in a project, simply click the Build checkbox of the root step).

**Save expand/collapse state of steps in project file:** If checked, the expand/collapse state of all

steps will be saved in the project file and used when the project file is reloaded (project files saved with this option enabled will override the following option).

*Note:* The user will not be prompted to save changes when closing a project or exiting the application when the only change is to the expand/collapse state of steps. You can manually persist these changes by explicitly saving the project before closing.

**Step levels to expand when opening projects:** Determines how many levels of child steps to expand when opening a project (the expand/collapse state of steps is not persisted with the project file).

**Clicking on step action icon toggles 'Include in Build' flag:** Determines if a step's Build flag is toggled when clicking on the step's icon. If unchecked, clicking on the icon will only select the step (the Build flag can still be toggled by clicking on the Build checkbox or by pressing the Space bar).

**Identify and use application icon for Run Program steps:** If checked, for Run Program steps whose command has an associated application icon, that icon will be used instead of the generic Run Program action icon.

**Use custom icon identifying macro type for Set Macro steps:** If checked, for Set Macro steps, the step icon will indicate the type of macro being updated

*Note:* The previous two settings will not take effect until another project is opened or created or the project is refreshed.

**Validate step properties:** If checked, required fields on the Step Properties dialog or pane will be validated (requiring a value to be provided) before changes are accepted.

**Auto-save project:** If a value is entered here, unsaved project changes will be saved at the specified interval.

### 2.7.17.3 Display

The Display tab of the user options dialog is used to configured display settings.

**Full row select:** Causes the entire grid row to be selected instead of the first column only

**Grid lines:** Displays grid lines in step, macro, and script tabs.

**Tree lines:** Displays connecting lines for items in the step and macro panes.

**Underline hot items:** Causes the item that the mouse hovers over in a grid to be underlined.

**Show macro tool tips:** Toggles the display of tool tips throughout the application.

**Max characters:** Specifies the maximum length of text to display in a tool tip (will truncate longer text and add ... to the end of tool tip). Windows may display a flickering tool tip with very long tool tip text, and this setting can be used to prevent flickering.

**Show tool tips while building:** Determines whether tool tips in the Step and Macros panes will be displayed during a build.

**Automatically refresh panes while building:** Determines whether the Macros, Call Stack, and Watches panes will automatically refresh during a build. If unchecked, the panes will only be refreshed when a build pauses, completes, fails, aborts, or is explicitly refreshed via *View | Refresh* on the menu.

**Sort Go to step dialog by step name:** If checked, steps in the Go to step dialog will be sorted

alphabetically. If unchecked, steps will be sorted in the order displayed in the step pane.

**Show only top-level steps in Go to step dialog:** If checked, only top-level (and 2nd or 3rd level if only 1 or 2 top-level steps) steps will be displayed in the Go to step dialog. If unchecked, all steps will be displayed.

**Format Watches pane values:** If checked, boolean properties in the Watches pane will be displayed as Yes/No rather than -1/0, a step's build status, rule comparison, and continue on failure properties will be formatted for display rather than showing the integer value.

#### 2.7.17.4 Columns

This tab of the user options dialog determines which columns are displayed in step tabs of the Step Panes. The mouse can also be used to drag and resize, show, and hide columns. The column order can be rearranged by dragging and dropping the column headers, and right-clicking on the column headers will also display a menu to toggle which columns are displayed (works in step, macro, and script tabs).

**Restore Defaults:** Restores the default displayed column selections.

#### 2.7.17.5 Build

This tab of the user options dialog is used to configure options that apply when a project is built.

**Save project before building:** Determines whether a project file is automatically saved before building.

**Reset build before building/starting from cursor or building step group:** If checked, when starting a build via Build From Cursor, Start From Cursor, or Build Step Group, the build status for all steps will be reset before the build is started.

**Display license or evaluation message at start of build:** When checked, the current license string or evaluation message is logged to the Output pane at the start of each build (useful as a reminder of the time remaining during evaluation).

**Use custom pane layout when building:** If checked, alternate pane layouts will be used for build and design mode.

**Show step pane for current step during build:** If checked, the step pane containing the current step will be shown when the current step changes during a build.

**Expand tree to show current step during build:** If checked, the step pane will be expanded to show the current step when the current step changes during a build.

**Scroll step pane to show current step during build:** If checked, the step pane containing the current step will be scrolled to show the current step when the current step changes during a build.

*Note:* The above the three options do not apply when single stepping a build; in this case, the current step will always be activate, expanded, and scrolled to. Also, if the user changes the active step pane or scrolls the current step pane while a build is active, the current pane/step will not be activated or scrolled to (until the build is canceled, fails or completes and is restarted, or the current step is navigated to [Go | Current Step on the menu bar]).

**Play a sound when build completes or fails:** If checked, a system sound is played when the build finishes; a different sound is played if the build fails. If unchecked, no sounds are played.

**Play a sound when stopping at a breakpoint:** If checked, a sound is played when the build stops at

a breakpoint.

**Honor command-line switches if build fails/canceled and continued:** If checked, and the build fails or is aborted and is then resumed, any command-line switches that affect the behavior of the build will be honored. If unchecked, the default interactive behavior will be restored (the command-line switches will be reset). For instance, if the /s flag is specified, breakpoints will not be stopped at and the app will close on completion. If this option is checked, breakpoints will continue to be skipped and the app will close when the build completes; if unchecked, when the build is restarted/continue, breakpoints will be stopped at and the app will not close when the build completes.

### 2.7.17.6 Output

This tab of the user options dialog is used to configure options that apply to the Output pane.

**Clear output pane before building:** When checked, the Output pane is cleared at the beginning of each build. If unchecked, the Output pane is appended to, and existing output from a previous build is not cleared.

**Automatically scroll output pane to show most recent output:** If checked, the Output pane will be scrolled during a build to show the most recent build output.

**Include timestamps:** If checked, a timestamp is displayed for each step logged to the Output pane.

**Include step numbers:** If checked, each step's index is also logged to the Output pane. This allows navigating between a step and its output to work more consistently even if multiple steps have the same name.

**Log skipped steps:** If checked, steps skipped during a build will be logged to the Output pane.

**Log level:** Determines which log messages will be logged to Output pane when building a project (messages with a level that is the same or lower than the level specified here will be logged). This can be useful to reduce the amount of build activity logged during a normal build. And if a step fails during the build, the FAILSTEP\_OUTPUT macro (available from Failure Steps) will contain all logged messages for that step, even if they were not logged due to the Log level setting.

**Logging of step starting events:** Determines the logging behavior of step starting build events to the Output pane.

**Maximum output to retain:** Specifies the maximum number of lines to keep in the Output pane. If the number of lines logged to the view exceeds this value, lines at the beginning will be removed. Enter 0 to prevent lines from being removed from the output pane.

*Note:* Even with this option checked, if the user scrolls the output pane up while a build is active, the output pane will not be automatically scrolled until the build is canceled, fails or completes and is restarted, or the user scrolls back to the end of the output pane.

**Font/size:** Specifies the font to use in the Output pane.

### 2.7.17.7 Colors

This tab of the user options dialog is used to configure the colors that are used in the Project and Output pane during a build. The color of in progress, successful, skipped and failed/aborted steps can be specified, and the background color for the current step when debugging can also be chosen.

### 2.7.17.8 Print/Backups

This tab of the user options dialog configures print / print preview and backup options.



**Print format:** Specifies whether the active pane or a detailed project report (including all step types and properties, macros, and project comments) will be printed.

**Include all macros and global subroutine steps in output:** If unchecked, only the project macros and steps are printed; if checked, all macros (temporary, project, global, and system) and any global subroutine steps are printed.

**Show extended step properties:** Determines whether custom step properties for each step are displayed. If unchecked, only the standard step properties are printed.

**Font:** Configures the font to use in the printed output.

**Create a backup file when saving projects:** If checked and a valid backup path entered, each time a project is saved, the previous copy of the project file is backed up. Any modified global application data files will also be backed up before saving (when a project is saved or Visual Build exits).

**Backup file directory:** The path to save backup files in.

**Number of backups to keep:** Specifies the number of recent backup copies to keep.

*Note:* Backup files are saved with a numeric (i.e., .1) extension. To use a backup file, remove the numeric extension from the filename and open the file or copy it to the desired location.

#### 2.7.17.9 Miscellaneous

The Miscellaneous tab of the user options dialog is used to configured additional settings.

**Auto complete folders and filenames in edit controls:** If checked, edit controls in the GUI will autocomplete from local file and folder names.

**Auto complete macros in Properties pane:** If checked, when editing properties in the Properties pane, typing %name can be used to display a list of macros that matches the typed value.

**Populate computer and service lists in new steps:** If checked, when new step actions with computer and service list fields are created, the drop-downs will be pre-populated with the available computer names and services. Loading this information can sometimes be slow and can be prevented by unchecking this option.

**Use GUI App for Tools | Create Scheduled Task:** If checked, the command used when creating a scheduled task from the Tool menu will specify the GUI App; if unchecked, the Console app executable will be specified.

**Confirm deletion of global steps and macros:** If checked, when deleting global subroutine steps or global macros, a prompt will be displayed to confirm the deletion. Deleting of global steps and macros is undoable, and the configuration files that global macros and steps are stored in are also backed up by default, but as an additional level of protection, the default behavior is to also prompt before deleting. This option can be unchecked to delete without confirming each time.

**Special characters to escape/unescape on copy/paste:** Determines which special characters are automatically unescaped/escaped when copying/pasting between Visual Build and other applications.

**File Viewer:** Defines the application that is used for viewing files when View is chosen from the:

- *Shell* -> View button in the Step Properties and Macro Properties dialogs
- The *View | Shell | View* menu for the selection or cursor position in the Step Grid, Output pane, or Properties pane. When a step is selected in the step grid, if the step's default property is a filename or folder, that filename or folder will be opened; when focus is in the Output pane, any selected filename or folder (or the filename and line number for the current cursor position if in the format

shown below) will be opened.

*Notes:*

- If the viewer application executable is not in the PATH environment variable, the full path must be specified.
- When a filename and line number is displayed in the Output pane in the format `C:\Path\To\Filename.ext(999) : optional error description` and if the configured viewer application supports a command-line flag for opening a file at a specific line number, the File Viewer supports a special syntax for providing the necessary parameters to the viewer. This is indicated by appending `|<additional flags>` to the File Viewer field, where `<additional flags>` contains any special parameters used to pass the filename and line number to the viewer application. The values `%1` and `%L` in the additional flags will be replaced by the filename and line number, respectively. Rather than selecting the filename, just click on or move the cursor to the line containing the filename and choose *View* from the right-click or *View/Shell* menus, or use the associated keyboard shortcut.

**Command Prompt:** Defines the program to use for opening a command prompt / console to when Command Prompt is chosen from the:

- *Shell -> Command Prompt* button in the Step Properties and Macro Properties dialogs
- The *View | Shell | Command Prompt* menu for the selection or cursor position in the Step Grid, Output pane, or Properties pane. When a step is selected in the step grid, if the step's default property is a filename or folder, that prompt will be opened to that folder; when focus is in the Output pane, the console will be opened to the selected folder in that pane.

## 3 Actions

Actions are a powerful feature of Visual Build. Every step has several standard properties. Each step also has an action type for performing a specific function within a build. Every action may have one or more input screens for entering the fields that are specific to that action, and it may optionally define a custom action component for performing the custom functionality of that action.

A new step for a given action is created by:

- Double-clicking an action in the Actions pane or dragging and dropping an action onto a step pane.
- Dragging and dropping a filename associated with an action from Explorer onto Visual Build. This creates the step and pre-populates the filename field.
- Choosing *Insert* from the *Edit* menu to insert a new step. This activates the Actions pane to select the action for the step.

Categories of pre-defined actions that are available:

- System
- Compression
- Embarcadero/Borland
- Files
- Help Authoring
- Installers
- Localization
- Microsoft
- Microsoft .NET
- Miscellaneous
- Network
- Server
- Test Driven Development
- Version Control

- Virtual Machines

All actions define a custom action component or script, which will be called when the step is built to perform the custom functionality for that action. The action will build the step and log output to the Output pane and log file if enabled.

Several advanced capabilities for actions are also available, including creation of user actions for plugging custom functionality into Visual Build.

*Note:* Some obsolete actions are no longer displayed by default (existing steps for these actions will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

## 3.1 System

### 3.1.1 Exit

The Exit action creates a step that jumps to the end of a build, exits a loop, or returns from a subroutine or failure steps.

**Exit Status:** The step build status to exit with:

- **Success:** If the Exit step is located in the main project steps, the build will complete without building any additional steps. If the Exit step is located in a call to a subroutine, the build will return from the subroutine without processing any additional steps, and any later steps after the Subroutine Call step will be built. If the Exit step is located in Failure steps, the build will return from the failure subroutine or failure steps without processing any additional steps.
- **Failure:** Causes the step and build (unless ignoring failure) to fail (if the step is configured to build failure steps, the failure steps will be built).
- **Cancel:** Causes the build to stop as if the user had stopped the build manually (failure steps are not built).

**Message:** A message to be logged before exiting (displayed in the Output pane and logged to a file if file logging is enabled).

**Exit loop or repeating build rule:** If checked, within a repeating Loop, Process Files group, or build rule, exits the loop instead of the entire build or subroutine.

*Note:* See the SingleInstance sample for an example of the Exit action.

### 3.1.2 Group

The Group custom action defines a step that does not perform an action itself, but can be used to group multiple child steps or to separate steps within a project. This action does not have any custom properties of its own. All the general step properties can be edited, and the step can define a conditional build rule that will apply to the step and its children.

### 3.1.3 Log Message

The Log Message action creates a step that logs a message to the Build output. The message will be displayed in the Output pane and will also be logged to a file if file logging is enabled.

**Message to log:** The message that will be logged (required).

**Level:** The message level.

### 3.1.4 Loop

The Loop action creates a step to repeat a set of child steps a specific number of times, once for each value in a list, a file, ADO recordset, MSXML node list, dictionary, or forever.

Create one or more child steps of any type (including Subroutine Call actions) and use the macros below to operate on each matching value. This action can be used with the Read File, Read INI, Read XML actions or any string or macro value to iterate over the contents of a file, a section of an INI file, matching values in an XPath query, a list of values, etc.

Each child step is built once for each value, and the following temporary macros are available (use in child steps to process each iteration value):

- **LOOP\_VALUE:** The loop value for the current iteration (empty string for *Forever* loop type).
- **LOOP\_COUNT:** The number of matching values for the last Loop action (-1 for *Forever* loop type).
- **LOOP\_INDEX:** The index of the current iteration (1-based).

**Type:** The type of loop to perform:

- *Count:* repeats the child steps the number of times specified in the Count field.
- *List of values:* repeats once for each value in the Value field.
- *File contents:* repeats for the contents of a file (using the specified *delimiter*).
- *ADO recordset:* repeats for each record in an ADO recordset object (the ADO recordset object will be stored in the LOOP\_VALUE macro).
- *MSXML node list:* repeats for each node in an MSXML node list (the MSXML node object for the current iteration will be stored in the LOOP\_VALUE macro).
- *Forever* repeats the child steps continuously (until aborted or exited with the Exit action or by using the step failure exit build/subroutine option).
- *Dictionary* repeats for each item in a Scripting.Dictionary object (the key for the current iteration will be stored in the LOOP\_VALUE macro).

**Value:** The count, string or array value, filename, or name of a macro containing the ADO recordset, MSXML node list, or Dictionary object to iterate over (required).

**Delimiter:** The delimiter to split the string for values to iterate over (optional; uses newline or each array element if blank). This action splits the given value into separate strings and then performs all child steps once for each value.

**Process empty values:** If checked, even empty values will be processed.

**Log value used for each iteration:** If checked, each value that is found will be logged.

*Notes:*

- See the Advanced, Files, Server, XML, and ContinuousIntegration samples for a demonstration of using the Loop action.
- This is an iterative action; if used with the Rebuild Selected build action or Test button, the values will be logged, but the child steps will not be processed; see the Methods of Building help topic for other ways to test iterating steps.
- A Loop action must be a child of a subroutine step (it can't be the subroutine entry point). Use a Group action as the subroutine entry point and the Loop step as a child of it.

### 3.1.5 Run Program

The Run Program action creates a step to launch any external application, program, batch file, or command script. Visual Build starts and monitors the application, captures any output and logs it to the Output pane (and a log file if enabled), and terminates the application if the build is stopped.

When the step completes, the following temporary macros are created or updated:

- **RUNPROGRAM\_EXITCODE:** If the Wait option is checked, the exit code of the process will be stored in this macro.
- **RUNPROGRAM\_PROCESSID:** If the Wait option is unchecked, the process ID of the launched process will be stored in this macro.

## Program Tab

## Input Tab

## Advanced Tab

## Remote Tab

*Note:* To run batch files (.bat), command scripts (.cmd), or individual shell commands, use the Batch File action. If using the Run Program action, the command should be prefixed with the DOSCMD system macro (i.e., %DOSCMD% call "C:\path\to\file.bat" arg1 "arg 2" or %DOSCMD% copy "%PROJDIR%\Test.exe" "\\server\deploy") so that the command is executed under a command shell/interpreter. Also, if you wish to execute multiple commands (& [run both], && [run second command if first succeeds], or || [run second command if first fails]), redirect output (> or >>), or use piping (|) on the output of an executable, the command should also be prefixed with DOSCMD, since the command interpreter is what implements these capabilities.

*Note:* To call the 64-bit version of cmd.exe when using the DOSCMD system macro, create a COMSPEC global macro with a value of %WINDIR%\Sysnative\cmd.exe.

### 3.1.5.1 Program Tab

This tab of the Run Program action configures information about the program or command to be run.

**Command:** Specifies the command to invoke. It can contain the executable name plus any parameters that should be passed to it (parameters that contain spaces must be surrounded in double quotes). If an absolute path is not specified and the executable is not found in the current path, the Windows directory, the Windows System directory, or the PATH environment variable, Visual Build will also look for the executable in the *App Paths* registry key.

*Note:* To run batch files (.bat), command scripts (.cmd), or individual shell commands, use the Batch File action. If using the Run Program action, the command should be prefixed with the DOSCMD system macro (i.e., %DOSCMD% call "C:\path\to\file.bat" arg1 "arg 2" or %DOSCMD% copy "%PROJDIR%\Test.exe" "\\server\deploy") so that the command is executed under a command shell/interpreter. Also, if you wish to execute multiple commands (& [run both], && [run second command if first succeeds], or || [run second command if first fails]), redirect output (> or >>), or use piping (|) on the output of an executable, the command should also be prefixed with DOSCMD, since the command interpreter is what implements these capabilities.

**Start In:** The path that will be the starting directory for the process (optional). Applications that use the working directory to locate files, such as NMAKE, require this parameter to be defined in order to work correctly. This setting is equivalent to setting the current directory in a Command/DOS Prompt via the CD command. Do not enclose the value in double-quotes.

*Advanced:* This field supports relative paths for nested steps: If the path is entered as a relative path, Visual Build will prefix the path of a "parent" step (one above it that has less indentation), and will do this recursively until there are no more parent steps or the path is no longer relative. For example, if the first step (indented one level) has a Start In path of c:\, the second step (indented two levels) a Start In path of *MyProject*, and the third step (indented three levels) a Start In path of *X*, Visual Build will use a Start In path of *c:MyProject* for step #2 and *c:MyProjectX* for step #3 (in this case, the fully

expanded path will be displayed in the field's tool tip).

**Read Output From:** Selects where to read the step's output from. Most processes write their output to standard output. Some (such as the Microsoft Visual Basic compiler) write their output to a file. If *A file* is selected, an Output File to read from must be entered, and Visual Build can delete this file before building the step if desired. Some processes may not have any output, and None can be chosen in this case, or if you do not want an application's output to be logged.

**Hide application window:** By default, the process window is hidden when building, but you can choose to show it with this field (Visual Build detects when Windows executables are called and always displays them with a window so that the user can respond if the application doesn't close as expected). This may be necessary for applications that might display a dialog box or message box. It can be useful when debugging a step that calls an application with a graphical interface.

*Note:* This option will be ignored for GUI applications if the Always show GUI applications option is checked.

**Success Exit Codes:** Visual Build determines the success of the process by examining the exit code of the process. By default, a zero (0) exit code is considered successful, and any other code is a failure. Sometimes, an application will return non-zero exit codes to indicate partial success or additional information, and multiple ranges of success exit codes can be specified in the format low1:hi1, low2:hi2, code3. For instance, 0:5, 10 would cause the values 0 through 5 and 10 to be considered successful exit codes.

*Note:* Some applications do not return an exit code; Visual Build can still invoke them but cannot directly determine success or failure of the step. In these situations, the system macro LASTSTEP\_OUTPUT can be used in the following step or the vbld\_StepDone step script event to determine failure (see the GetProjVer.bld sample for more details).

### 3.1.5.2 Input Tab

This tab of the Run Program action specifies any console inputs to send to the program being run.

**Provide standard input from:** Specifies a file or string to provide to the program's standard input. This can be used for programs that read input from standard input when run in an automated fashion. If none is specified, no input will be sent to the program's standard input. Otherwise, the contents of the given file or the string entered will be passed to standard input when the program is run.

### 3.1.5.3 Advanced Tab

This tab of the Run Program, Batch File, PowerShell, VisBuildPro Project, and other Run Program-derived actions is used to specify advanced options for the program that is called.

**Username:** Specifies the user account that the program will run under (optional). If blank, the user identity will be inherited from the parent build instance. To specify a domain, use the format user@domain.

**Password:** Specifies the password of the user account (optional).

**Options:** Specifies a logon option (optional, applies only if credentials are supplied).

*Logon with profile:* Log on, then load the user profile in the HKEY\_USERS registry key. Loading the profile can be time-consuming, so it is best to use this only if you must access the information in the HKEY\_CURRENT\_USER registry key. If this option is not specified, access to information in the HKEY\_CURRENT\_USER registry key may not produce results that are consistent with a normal interactive logon.

*Network credentials only:* Log on, but use the specified credentials on the network only. The new

process uses the same token as the caller, but the system creates a new logon session within LSA, and the process uses the specified credentials as the default credentials. This value can be used to create a process that uses a different set of credentials locally than it does remotely. This is useful in inter-domain scenarios where there is no trust relationship. The system does not validate the specified credentials. Therefore, the process can start, but it may not have access to network resources.

**Batch:** Intended for batch servers, where processes may be executing on behalf of a user without their direct intervention. This type is also for higher performance servers that process many plaintext authentication attempts at a time, such as mail or web servers. Only applies if the UseLogonCreateProcessAsUser property is True.

**Service:** Indicates a service-type logon. The account provided must have the service privilege enabled. Only applies if the UseLogonCreateProcessAsUser property is True.

**Priority:** Specifies the priority that the process will run at.

**Affinity:** Indicates the processor affinity for the threads of the process.

**Wait for completion:** When checked, Visual Build waits for the called program to finish and exit before continuing, displaying any step output in the Output pane and log file. Uncheck this option to continue to the next step without waiting for the current step to complete. No output from the step will be captured, and if the process is started, the step will be treated as successful.

**Notes:**

- When this option is unchecked, the process ID of the launched process will be stored in the **RUNPROGRAM\_PROCESSID** temporary macro.
- When this option is checked, the exit code of the process will be stored in the **RUNPROGRAM\_EXITCODE** temporary macro.

**Log the command-line that is used:** If checked, the command used to call the external program will be logged when the step is built. Can be useful for debugging purposes.

**Use interactive desktop:** If unchecked, the process will inherit the desktop and window station of the calling process (permission for the specified user account to the inherited window station and desktop are added automatically). If checked, the interactive windowstation and desktop will be used for the created process.

**Note:** Permissions for the specified user account are not automatically added to the interactive window station and desktop when checked, and GUI applications will not have full access to draw properly unless the necessary permissions are assigned manually (see here for more details).

**Command to run before main command:** Specifies command(s) to run before running the action's main command (optional). This can be useful for executing a batch file or command script that creates or updates environment variables (for instance, the Visual C++ `vsvars32.bat` or DDK `setenv.bat`) -- the main command's environment will inherit the updated environment variables set in batch file. Multiple commands can be performed by separating each command with the `&` character. If a command is specified here, the main command will also run under the command shell ( `DOSCMD`), allowing piping ( `|`) and redirection ( `>` or `>>`) for commands that are not shelled by default. To force the main command to run under the command shell *without* running another command first, enter `%DOSCMD%` in this field.

**Note:** To call the 64-bit version of `cmd.exe` when running the command, create a `COMSPEC` global macro with a value of `%WINDIR%\Sysnative\cmd.exe`.

**Prefix main command:** If checked, rather than running *Command to run before main command* as a separate command before the main command (separated from the main command by `&&`), the command will instead be prefixed to the main command (the command can also be suffixed with

another symbol to effect that behavior [i.e., &, |]). When checked, the command will not automatically run under the command shell (begin the command with %DOSCMD% to do so).

#### 3.1.5.4 Remote Tab

This tab of the Run Program, Batch File, PowerShell, VisBuildPro Project, and other Run Program-derived actions is used to execute the command on a remote computer.

**Computer:** The name of one or more remote computers (separated by commas) to run the command on. If blank, the program will be executed on the local computer.

**Timeout:** Specifies a timeout (in seconds) for connecting to remote computers (optional).

**Run in System account:** If checked, the remote process will be run in the System account, the account that Windows services and Windows processes such as Winlogon and the Local Security Authority Subsystem Service (LSASS) run in.

*Note:* For remote execution, if a username is not provided on the Advanced tab and *Run in System account* is unchecked, the remote system impersonates the account that Visual Build is running under on the local computer, and the remote process will not have access to network resources that the user normally would have access to. If the account that Visual Build is running in doesn't have local administrator privileges on the remote system, the process needs access to network resources, or to run the process under a different account, provide a username and password of another user that the remote process should run under.

**Don't load user profile:** If checked, the specified user's account profile will not be loaded.

**Interact with desktop session:** Interact with the desktop of the specified session on the remote system (if checked and no session is specified, the process runs in the console session).

**Run with elevation:** If the target system is Vista or later, the process will run with the account's elevated token, if available. *Note:* This option requires PsExec v1.98 or later.

**Run as limited user:** Strips the Administrators group and allows only privileges assigned to the Users group.

**Display UI on Winlogon secure desktop:** Applies only when %COMPUTERNAME% is used in the Computer field.

**Copy program to remote system:** If checked, the program specified in the Command field will be copied to the Windows directory of the remote system and deleted after the program finishes running. Otherwise, the program specified in the Command field must be available in the PATH environment variable of the remote system or the full path to the program (on the remote computer) must be specified.

**Copy only if version number higher than on remote system:** If checked, the program will only be copied if its version number is higher than the version on the remote system.

**Copy even if file already exists on remote system:** If checked, the program will be copied even if it already exists on the remote system.

**Log PsExec output:** If checked, all output from PsExec.exe will also be included in the build output.

**Remote service name:** Specifies the name PsExec assigns to its remote service (optional). This can improve performance when multiple users are interacting concurrently with a system, since each will have a dedicated PsExec service. *Note:* This option requires PsExec v2.0 or later.



**Additional PsExec options:** Specifies additional command-line arguments for PsExec.

*Notes:*

- Use of this tab requires that the [SysInternals PsExec](#) utility be available in a path of the PATH environment variable on the local computer (alternatively, PAExec can be used by overriding the PsExecCmd property). See the Server.bld sample for sample steps to download PsExec and install it in the System path if not found (Visual Build must be run as administrator on Windows Vista and later in order to build). See this SysInternals forum post for setup of remote execution.
- Paths and filenames entered in the various fields must be valid paths and filenames relative to the remote computer.
- To run a program with a graphic interface (GUI) remotely, uncheck the *Hide application window* checkbox on the Program tab.
- When using PsExec v1.58 or later, if *Wait for completion* on the Advanced tab is unchecked and the process was successfully started on the remote computer, the process ID of the remote process will be stored in the RUNPROGRAM\_PROCESSID temporary macro.
- When calling PowerShell remotely, you must set the *Input format* field on the Options tab to *None* to work around a known issue that keeps PowerShell from exiting when the script finishes.

### 3.1.6 Run Script

This action creates a step to execute script code from a project step. The language can be selected from the list, or the ProgId of any registered script engine can be specified. The script code is edited in the main dialog text box. The script code can also call any script code for that language defined in the Script Editor. Script code in the Code field should not be enclosed in brackets (script expressions in other fields must be enclosed in brackets to indicate script code).

When the step is built, Visual Build will instantiate the script engine, add all the script for that language from the Script Editor into the engine, and execute the code in the Code field. The script code can access the Visual Build automation objects via the Application, Project, Step, and Builder global items, any global script code defined for that language, create other COM objects, etc. Any output or errors will be logged to the Output pane and log file (if enabled). The step's BuildStatus property can be set to *vbldStepStatFailed* in the script code to signal failure of the step.

*Note:*

- Like other step fields, the Code field can also reference macros which will be expanded when building. However, unlike other fields that contain macros, a tool tip is not displayed to show the expanded value unless some text is selected. To view an expanded value, select the text containing the macro(s), move the mouse cursor out of the field and back in, and the expanded tool tip will be displayed.
- Macro references (i.e., %MACRO\_NAME%) are expanded **before** the script is fed to the script engine, so script code that creates or updates a macro's value should not also reference that macro via %MACRO\_NAME%; instead, use the macro variable directly (i.e., objMacro.Value) or expand the macro using `Application.ExpandMacrosAndScript("%%MACRO_NAME%%")`, which delays evaluation of the macro value to that point in the code rather than before the script code is executed.
- To create a temporary macro in script code (VBScript), use

```
Set objMacro = vbld_TempMacros.Add("MACRO_NAME", "macro value")
or
Set objMacro = Application.Macros(vbldMacroTemporary).Add("MACRO_NAME", "macro value")
```

- The Step.BuildStatus property can be updated to change the success/failure status of the step. If the step is marked to continue on failure, throwing or raising an error from script code will not ignore the failure (providing a way to abort the build even when the step is marked to continue on failure).

*Note:* The Script.bld sample demonstrates creating or incrementing a macro value.

### 3.1.7 Set Macro

This action creates a step which sets or deletes a Visual Build macro. When the step is built, the specified macro will be created/updated or deleted.

**Name:** The name of the macro to update.

**Delete the macro:** If checked, the macro will be deleted. If the macro does not exist, the step will do nothing.

**Value:** The value to assign to the macro.

*Notes:*

- Bracket characters [ and ] normally denote a script expression to be inserted into a field; to insert literal brackets, use two bracket characters [ [ or ] ].
- The percent sign character % is normally used around a macro name for its value to be expanded within the field; to insert a literal percent character, use two percent characters %%
- Ctrl+C copies the value without expanding macros or script. To copy the value with all macros expanded, type Ctrl+M. Ctrl+Shift+M copies with macros and expanded and with script expressions evaluated.
- Ctrl+Enter inserts a new line into the edit control.

**Description:** The description to assign to the macro.

**Category:** A category to group the macro in when displayed in the macros pane.

**Don't expand macros and script:** If checked, any macros, script expressions, or double (literal [ , ], and/or %) characters in the Value field will not be evaluated at the time the macro is set; instead they be evaluated whenever the new macro is referenced during the build.

**Add to environment variables:** Determines whether the macro will be added/updated (or removed if deleting) as an environment variable for the current build process and external programs that are run after the macro has been created/updated or deleted.

**Encrypt macro value:** Determines whether the macro value will be encrypted when displayed and when stored in the project file.

**Macro Type:** Specified the type of macro that will be created.

*Note:* See the Advanced, Prompt, and Logging samples for examples of the Set Macro action.

### 3.1.8 Subroutine Call

Creates a step to call a project or global subroutine. Subroutines provide a way to modularize build projects.

**Subroutine to call:** Name of the subroutine step to call (required). This can be any step at the root level on the Subroutine steps or Global subroutine steps tab. The drop-down list is populated with the list of available subroutine names. Names are matched without case sensitivity.

**Parameter Name:** Names of parameters to pass to the subroutine. A temporary macro will be created for each parameter when the subroutine is called and can be used anywhere within the subroutine. Subroutine steps can reference any macro name, not just those defined in this list, but this provides a convenient way to quickly define macros to pass to the subroutine steps. When a subroutine returns, all the temporary macros defined for calling that subroutine will be popped (deleted if there was previously no macro with the name or updated to its original value if it did exist previously).

**Parameter Value:** Values of parameters to pass to the subroutine.

Enter a macro name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Expand macros in parameters:** If checked, any macros referenced in the value fields (on both tabs) will be expanded before the subroutine is called. If unchecked, the macros will not be expanded until the parameter macro is referenced within the subroutine.

*Note:* See the Advanced and XML samples for examples of the Subroutine Call action.

### 3.1.9 VisBuildPro Project

Use this custom action to create a step to call or chain to another Visual Build project within a build.

When the step is built, the appropriate GUI or Console command-line command is generated and passed to the Builder object's RunProgramEx automation method. The generated command-line will be logged to the Output pane if the Log option is checked. This action is also a handy way to set up automated build commands. Configure the step as needed, build it, and click the Create Scheduled Task button to create a scheduled task for the build.

#### Project tab

#### Parameters tab

#### Options tab

#### Advanced Tab

#### Remote Tab

*Note:* See the Chain.bld sample for a project utilizing this action.

#### 3.1.9.1 Project Tab

This tab of the VisBuildPro Project action specifies the project to build and launch settings.

**Filename:** The filename of the project to build (required).

*Note:* When a VisBuildPro Project action is selected in a Step pane, *View / Shell* on menu (or right-click menu) can be used to launch the chained project in Visual Build.

**Profile:** Specifies a build profile for the build (optional). Only the steps with a profile matching the build profile (or steps with a blank profile having an ancestor step whose profile matches) will be included in the build.

**Mode:** Which command-line app to launch. If *Match build context* is selected, if the step is built from the GUI App, another instance of the GUI App will be launched; if built from the Console App, the Console App will be launched to build the chained project. If the GUI App is launched and the build is single-stepped (F10) for the action, the child project will be started with debugging (single-step) as well.

**Run GUI App in silent mode:** Adds the /s switch to the GUI App command-line.

*Note:* Check this option and uncheck *Don't close GUI App if build fails* to cause the chained instance to exit on build failure.

**Pause at first step for debugging:** Adds the /d switch to the GUI App command-line. When chaining to a project on the local computer, this flag will also be added automatically if the VisBuildPro Project step is single-stepped.

**Don't close GUI App if build fails:** Adds the /f switch to the GUI App command-line. Normally, when silent mode is specified, the chained app closes on a failure; this option can be used to override that behavior.

**Capture output:** If checked, any output that the chained Visual Build instance writes to standard output will be captured and logged.

*Notes:*

- The console app writes its output to standard output, but the GUI app does not (since it is a Windows application), so output can only be capture from the console app.
- This option does not apply if the *Wait for completion* option on the Advanced tab is unchecked.

**Password:** Specifies the project password used for encrypted protected properties (optional).

**Create Scheduled Task:** Click this button to automatically create a Scheduled Task using all the settings configured for the step. Any changes to the step will be applied before the task is created.

### 3.1.9.2 Parameters Tab

This tab of the VisBuildPro Project action specifies macro values to pass to the chained project.

**Macros:** Allows the entry of macros to assign as temporary macros when calling the chained project. Macros entered in the value field will be expanded before passing to the chained project (enter double percent signs if the macro should be evaluated within the chained project instead).

Enter macro name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Also pass through macros provided on the command-line:** If checked, any macros passed to the current instance of Visual Build on the command-line will be passed to the chained project.

**Install path:** Used to specify the installation path of Visual Build (on the remote computer) if it is installed to a different path than on the local computer (optional).

**Options:** Specify any additional command-line options or additional macros to pass.

### 3.1.9.3 Options Tab

This tab of the VisBuildPro Project action is used specify additional options.

**Log file:** Specifies a filename to use for file logging (optional). Sets a temporary LOGFILE macro to override any global or project-level LOGFILE macro.

**Configuration files path:** Specifies an alternate path for loading of all configuration files (optional). If this path is specified, the remaining options do not apply since this setting overrides paths for all configuration files.

**Global macros file:** Specifies an alternate location for the global macros file (optional).

**Global scripts file:** Specifies an alternate location for the global scripts file (optional).

**Global subroutine steps file:** Specifies an alternate location for the global subroutine steps file (optional).

**Application options file:** Specifies an alternate location for the application options configuration file (optional).

**Start in:** The path that will be the starting directory for the child process (optional).

## 3.2 Compression

### 3.2.1 7-Zip

The 7-Zip action creates a step to create, add, update, or test files in a 7-Zip, ZIP, GZIP, BZIP2, or TAR file or extract or list files in a 7-Zip, ZIP, GZIP, BZIP2, TAR, RAR, CAB, ISO, ARJ, LZH, CHM, MSI, WIM, Z, CPIO, RPM, DEB or NSIS file.

This action requires 7-Zip to be installed. It has been tested with 7-Zip v4, v9, and v15 and may work with other versions as well.

#### 7-Zip Tab

##### Options tab

##### Advanced Tab

##### Remote Tab

*Notes:*

- The ZIP Files and UNZIP Files actions can also be used to create, update, and extract from ZIP, GZIP, and TAR files.
- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.2.1.1 7-Zip Tab

This tab of the 7-Zip action specifies the output file and the input files to be added or updated.

**Filename:** The name of the compressed file to process (required).

**Action:** The action to perform on the file:

- *Add* always adds any matching files to the compressed file (if the output file does not already exist, it will be created).
- *Update* adds any new files to the compressed file and replaces any existing items that have been modified more recently than their corresponding version in the output file (if the file does not already exist, it is created).
- *Extract* extracts files from the compressed file to the Output folder (or current directory if not specified).
- *Test* tests the archive file.
- *List* lists files in the compressed file.
- *Delete* removes files and/or folders from an existing compressed file.

**Type:** Specifies the type of archive (optional, defaults to 7z format).

**Include:** Specifies the files or wildcards to compress (optional, defaults to files in working directory if not specified). One or more absolute or relative paths and filenames, with wildcards (\* and ?), can be specified, each on a separate line.

**Exclude:** Specifies the files or wildcards to exclude (optional). One or more absolute or relative paths and filenames, with wildcards (\* and ?), can be specified, each on a separate line.

**Include subfolders:** Determines if subdirectories will be searched recursively for matching files.

**Don't save path information for files in subfolders:** If checked, the folder information for recursed folders is not used (applies only when extracting).

**Working folder:** The path to search for files when adding or updating or extract compressed files to when extracting (optional).

**Password:** Password to encrypt or unencrypt the compressed file with (optional).

### 3.2.1.2 Options Tab

This tab of the 7-Zip action specifies the additional options about the compression operation.

**Answer Yes to all prompts:** If checked, overwrite queries will be suppressed and files on disk with same filenames as in archive will be overwritten.

**Enable Large Pages mode:** Large Pages mode can increase speed of compressing, but this can result in a delay at start of compressing for allocating large pages. Also Task Manager doesn't show real memory usage of program when using large pages. This feature works only on Windows 2003 / XP x64 / Vista / 7, and Visual Build must be running with administrative privileges. Recommended size of RAM: 1 GB or more. To install this feature you must run 7-Zip File Manager at least once, close it and reboot system.

**Delete existing output file before starting:** If checked and the output file exists, it is deleted before the compression operation starts (use only with Add or Update actions).

**Show technical information when listing:** If checked, detailed information will be included for the List action.

**Create self-extracting archive:** If checked, creates a self extracting archive executable. The SFX module can be specified in the next field (optional; the standard console SFX module will be used if not specified).

**Temp folder:** Sets working directory for temporary base archive (optional, uses the same directory as the old base archive file if not specified).

**Additional command-line options:** Use this field to enter any additional 7z command-line options to be added to the call that is constructed (see Command Line Version in the 7-Zip help file for details on available options). Use this to supply flags that are not explicitly supported via the action fields.

**Override the 7-Zip executable filename:** If not specified, the action locates `7z.exe` automatically. If the action is unable to locate the command-line tool or multiple versions are installed, enter the full drive+path+filename to `7z.exe` here.

### 3.2.2 Enhanced Unzip Files

The Enhanced Unzip Files action creates a step to extract files from a ZIP file, with additional options. When built, the action finds all matching files and folders in the given archive file (recursively searching for matches if specified) and extracts to the folder specified. When the step completes, the following temporary macros are created or updated:

- **UNZIP\_PROCESS\_COUNT** = The number of files extracted from the ZIP file

## Unzip tab

## Options tab

## Attributes tab

### Notes:

- See the Recurse sample for an example of this action.
- The Enhanced ZIP Files action can be used to create or update ZIP files.

### 3.2.2.1 Unzip Tab

This tab of the Enhanced Unzip Files action specifies options for unzipping.

**Filename:** The source file to extract files from (required).

**Action:** The action to perform. *Extract* always extracts any matching files; *Update* extracts only files that don't exist in the destination or that are older than the corresponding version in the archive file; *Freshen* extracts only existing files that are older than the version in the archive file.

**Files to process:** Used to specify files or folders to include or exclude when unzipping (optional). Specific attributes can also be included or excluded on the Attributes tab.

**Include subfolders:** Determines if subdirectories in the archive file will be searched recursively for matching files.

**Include empty subfolders:** If checked, matching empty subdirectories will be created in the archive.

**Use directory names:** If checked (the default), the folder information in the archive file will be used when extracting files. If unchecked, all files will be extracted to the destination directory.

**Destination:** The destination path to extract files to (optional, will use the current directory if not specified). **Important:** *Any existing, writeable files (and read-only files if the next option is checked) in the destination folder may be overwritten when extracting.*

**Overwrite read-only files:** If checked, read-only files in the destination will be replaced. If unchecked and a read-only file to be replaced exists, the file will be skipped.

### 3.2.2.2 Options Tab

This tab of the Enhanced Unzip Files action specifies additional options.

**Password:** Password to decrypt the files in the archive file (optional). This action can unencrypt ZIP files encrypted using Zip 2.0 password protection or Single Password Advanced Encryption Standard (AES) strong encryption (reading of WinZip version 9.0 AES-1 and AES-2 encrypted items using 128-, 192- and 256-bit key lengths is also supported).

**Continue on warnings:** If checked, the ZIP operation continues processing on all warning conditions.

**Restore security descriptors:** Indicates whether to restore any NTFS file security descriptors present in the archive.

**Log files that are processed:** If checked, any files extracted from the archive file will be logged.

**Log progress:** If non-zero value is provided here, the file progress (if logging files) or overall progress (if not logging files) is logged every x percent.

### 3.2.3 Enhanced Zip Files

The Enhanced Zip Files action creates a step to create, add, or update files in a ZIP or SFX (self-extracting executable) file, with additional options. When built, the action finds all matching files and folders (recursively searching for matches if specified) and compresses them into the specified ZIP or EXE file. When the step completes, the following temporary macros are created or updated:

- **ZIP\_PROCESS\_COUNT** = The number of files processed in the ZIP file

#### Zip tab

#### Compression tab

#### Options tab

#### SFX tab

#### Attributes tab

##### Notes:

- See the Recurse sample for an example of this action.
- The Enhanced Unzip Files action can be used to extract files from ZIP files.

#### 3.2.3.1 Zip Tab

This tab of the Enhanced Zip Files action specifies the output file and the input files to be added or updated.

**Action:** The action to perform on the file:

- *Add* always adds any matching files (if the output file does not already exist, it will be created).
- *Update* adds any new files to the output file and replaces any existing items that have been modified more recently than their corresponding version in the output file (if the file does not already exist, it is created).
- *Freshen* replaces only existing items in the output file, and only those that were modified more recently than the version in the output file (the output file is not created if it doesn't exist).
- *List* lists files in the ZIP file.
- *Test* tests files in the ZIP file.
- *Delete* removes files and/or folders from an existing ZIP file.
- *Convert ZIP to SFX* converts a ZIP file to SFX (self-extracting EXE).
- *Convert SFX to ZIP* converts a SFX to a ZIP file.

**Source folder:** The source directory containing the files to add to the compressed file (optional, will use the current directory if not specified). If converting to ZIP to SFX, specify the ZIP filename here. If converting to SFX to ZIP, specify the EXE filename here.

**Files to process:** Used to specify files or folders to include or exclude when zipping (optional).

**Include subfolders:** Determines if subdirectories will be searched recursively for matching files.

**Don't include path information for files in subfolders:** If checked, the folder information for recursed folders is not stored in the output file.

**Include empty subfolders:** If checked, matching empty subdirectories will be created in the archive.

**Save full path info:** If checked, the full path of each file is stored in the compressed file. If unchecked (the default), only relative paths of files in subfolders are stored in the output file.



**Output file:** The output file to add/update the files to (required). To create a ZIP file, specify a filename with ZIP extension; to create a SFX (self-extracting executable), enter a filename with EXE extension. If converting ZIP to SFX, enter the EXE filename here. If converting SFX to ZIP, enter the ZIP filename here.

### 3.2.3.2 Compression Tab

This tab of the Enhanced Zip Files action specifies compression and encryption options.

**Compression method:** The compression method to use. For *JPEG* method, the files to compress must be valid JPEG files. For *WAV* method, files must be valid WAV files. *Best* method automatically selects the most appropriate compression method for compressing the file (JPEG files are compressed using JPEG recompression, WAV sound files are compressed using WavPack, and remaining file types are compressed using deflate).

*Note:* Some older ZIP utilities cannot read files compressed with methods other than Deflate.

**Compression factor:** A numeric value indicating the level of compression (required). Can be a value between 0 (no compression) through 9 (maximum compression).

**Password:** Password to encrypt the compressed files in the ZIP file with. The contents of each file in the output file will be encrypted.

**Encryption method:** Specifies the encryption method to use:

- PKZIP 2.0-compatible: Uses standard encryption to secure the file using PKZip version 2.0 encryption.
- 128-, 192-, or 256-bit AES: Uses strong encryption to secure the .zip file using the PKZip version 5.1 Single Password AES Encryption format using 128, 192 and 256 bit key lengths.

**Temp folder:** Specifies the folder to write the temporary ZIP file to.

**Comment:** Specifies a comment to be stored in the output file (optional).

### 3.2.3.3 Options Tab

This tab of the Enhanced Zip Files action specifies additional options.

**Delete existing output file before starting:** If checked and the output file exists, it is deleted before the compression operation starts.

**Skip locked files:** If checked, causes the action to exclude any files that are currently locked or in exclusive use by another process.

**Continue on warnings:** If checked, the ZIP operation continues processing on all warning conditions.

**UTF-8 encoding of filenames:** If checked, Unicode filenames are stored in UTF-8 in the traditional filename fields of the central and local header (maximizes compatibility with modern zip utilities). If unchecked, Unicode filenames are stored in UTF-8 in extra fields and dummy ANSI values are placed in the traditional filename fields of the central and local header (maximizes compatibility with older zip utilities that may not support Unicode filenames).

**Store security descriptors:** If checked, stores NTFS file security descriptors when zipping (and restores if possible when unzipping).

**Reset archive attribute:** If checked, after adding/updating a file, the archive attribute of the source file is cleared.

**Log files that are processed:** If checked, any files added or updated to the output file are logged.

**Log progress:** If non-zero value is provided here, the file progress (if logging files) or overall progress (if not logging files) is logged every x percent.

#### 3.2.3.4 SFX Tab

This tab of the Enhanced Zip Files action specifies self-extracting executable (SFX) options.

**Note:** To compress existing files and create a SFX, enter the EXE filename in the *Output file* field on the Zip tab. To convert an existing ZIP file to SFX, specify an existing ZIP filename in the *Source folder* field and the EXE filename in the *Output file* field.

**Show unzip options:** If checked, displays default settings and allows modification before unzipping. If unchecked, uses default options specified below.

**Show prompts:** If checked, shows prompts for password, file overwriting and more.

**Show operation results:** If checked, shows results of the unzip operation.

**Show progress bar:** If checked, displays a progress bar while unzipping.

**Show about box:** If checked, displays an About button on the options dialog.

**Require elevation:** If checked, requires elevated administrator privileges on Windows Vista and higher (and Windows will display a UAC prompt if run from a non-elevated session).

**Existing file behavior:** Determines how to treat files that already exist in the destination.

**Destination folder:** Specifies the default destination to extract the files to (optional). Can include environment variables such as %TEMP%.

**Default password:** Specifies the default password for decrypting.

**Startup message:** Specifies a message to be displayed when the SFX is run.

**About message:** Specifies a message to display in the About box.

**ReadMe filename:** Specifies the filename of a text file to display after extracting. Can include environment variables such as %TEMP%.

**Command to run:** Specifies a command to run after extracting the files. Can include environment variables such as %TEMP%.

#### 3.2.4 UNZIP Files

The UNZIP Files action creates a step to extract files from a ZIP, GZ (gzip), or TAR file. When built, the action finds all matching files and folders in the given archive file (recursively searching for matches if specified) and extracts to the folder specified. When the step completes, the following temporary macros are created or updated:

- **UNZIP\_PROCESS\_COUNT** = The number of files extracted from the archive file

*Notes:*

- The ZIP Files action can be used to create or update archive files.
- The Enhanced Unzip Files action can be used for advanced options.
- This action is not available in the 64-bit edition of Visual Build.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will

build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

**Filename:** The source file to extract files from (required). Can be a ZIP, GZ (gzip), or TAR file.

**Action:** The action to perform. *Extract* always extracts any matching files; *Update* extracts only files that don't exist in the destination or that are older than the corresponding version in the archive file; *Freshen* extracts only existing files that are older than the version in the archive file.

**Also extract files from TAR:** When extracting a GZ file containing a TAR file, checking this option will first decompress the TAR file, and then extract the files from the TAR archive to the destination folder. If unchecked, the file compressed in the GZ file will be extracted to the destination folder.

**Files to process:** Used to specify files or folders to specifically include (optional). If the Include field is blank, all file and folder names in the archive file will be included. To include specific files, add each mask on a separate line in the Include field. Masks can be any valid file mask (for instance \*.cpp, abc\*, xyz\*.\*, myfile.rc?, etc.). To include folders, prefix the line in the Include field with the folder name(s) (i.e., abc\xyz\\*.txt).

*Note:* The Enhanced Unzip Files action can be used for enhanced filtering options.

**Include subfolders:** Determines if subdirectories in the archive file will be searched recursively for matching files.

**Use directory names:** If checked (the default), the folder information in the archive file will be used when extracting files. If unchecked, all files will be extracted to the destination directory.

**Dest folder:** The destination path to extract files to (required). **Important:** *Any existing, writeable files (and read-only files if the next option is checked) in the destination folder may be overwritten when extracting.*

**Overwrite read-only files:** If checked, read-only files in the destination will be replaced. If unchecked and a read-only file to be replaced exists, the step will fail.

**Log files that are processed:** If checked, any files extracted from the archive file will be logged.

**Convert long filenames to 8.3 format:** If checked and if any of the extracted files have long filenames in their path, those filenames are modified to create short filenames in the ZIP file by removing any spaces and taking the first characters (8 for folders/files and 3 for extensions) of each file component. For example, the filename `my new folder\my long filename.extension` would be changed to `mynewfol\mylongfi.ext`.

**Do not convert filenames to OEM character set:** If unchecked, the action will convert filenames in the ZIP file to their OEM character set. This may be necessary to handle ZIP files created by old ZIP tools such as PKZIP 2.04g and files created by the ZIP Files action with the *Generate PKZIP 2.04g compatible file* option checked. When unzipping gzip files, checking this option will cause long filenames to be ignored.

**Password:** Password to decrypt the files in the archive file (optional). This action can unencrypt ZIP files encrypted using Zip 2.0 password protection or Single Password Advanced Encryption Standard (AES) strong encryption (reading of WinZip version 9.0 AES-1 and AES-2 encrypted items using 128-, 192- and 256-bit key lengths is also supported), as well as encrypted gzip files.

### 3.2.5 ZIP Files

The ZIP Files action creates a step to create, add, or update files in a ZIP or GZ (gzip) file. When built, the action finds all matching files and folders (recursively searching for matches if specified) and

compresses them into the specified file. When the step completes, the following temporary macros are created or updated:

- **ZIP\_PROCESS\_COUNT** = The number of files added or updated in the compressed file

## ZIP tab

### Options tab

*Notes:*

- The UNZIP Files action can be used to extract files from ZIP, GZIP, and TAR archives.
- The Enhanced Zip Files action can be used for advanced options.
- This action is not available in the 64-bit edition of Visual Build.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.2.5.1 ZIP Tab

This tab of the ZIP Files action specifies the output file and the input files to be added or updated.

**Action:** The action to perform on the file (Update and Freshen apply only for ZIP files):

- *Add* always adds any matching files (if the output file does not already exist, it will be created).
- *Update* adds any new files to the output file and replaces any existing items that have been modified more recently than their corresponding version in the output file (if the file does not already exist, it is created).
- *Freshen* replaces only existing items in the output file, and only those that were modified more recently than the version in the output file (the output file is not created if it doesn't exist).
- *Delete* removes files and/or folders from an existing compressed file.

**Source folder:** The source directory containing the files to add to the ZIP file (optional, will use the current directory if not specified).

**Files to process:** Used to specify files or folders to specifically include or exclude (optional). If the Include and Exclude fields are blank, all file and folder names in the Source folder will be included. To include or exclude files, add each mask on a separate line in the Include and/or Exclude fields. Masks can be any valid file mask (for instance \*.cpp, abc\*, xyz\* \*, myfile.rc?, etc.). To include or exclude folders, prefix the line in the Include/Exclude field with the folder name(s) (i.e., abc\xyz\*.txt).

*Note:* The Enhanced Zip Files action can be used for enhanced filtering options.

**Include subfolders:** Determines if subdirectories will be searched recursively for matching files.

*Note:* The Delete action is always recursive.

**Don't save path information for files in subfolders:** If checked, the folder information for recursed folders is not stored in the output file.

**Save full path info:** If checked, the full path of each file is stored in the compressed file. If unchecked (the default), only relative paths of files in subfolders are stored in the output file.

**Output file:** The output file to add/update the files to (required). Can be a ZIP, GZ (gzip), or TAR file. When creating gzip files, the matching files will first be stored in a temporary TAR file, and then the TAR file will be compressed into a GZ file.

**Password:** Password to encrypt the compressed files in the output file with. The contents of each file in the output file will be encrypted.

**Encryption method:** Specifies the encryption method to use (applies only to ZIP files):

- **PKZIP 2.0-compatible:** Uses standard encryption to secure the file using PKZip version 2.0 encryption.
- **128-, 192-, or 256-bit AES:** Uses strong encryption to secure the .zip file using the PKZip version 5.1 Single Password AES Encryption format using 128, 192 and 256 bit key lengths.

### 3.2.5.2 Options Tab

This tab of the ZIP Files action specifies the additional options about the compression operation.

**Include only if archive attribute is set:** If checked, only files with the Archive attribute set are included.

**Reset the archive attribute:** If checked, after adding/updating a file, the archive attribute is cleared in the source file.

**Include hidden and system files:** Determines whether files with the system or hidden attribute will be included.

**Compression factor:** A numeric value indicating the level of compression (required). 0 or 10 = no compression (fastest method since files are simply stored); 1-9 = compression via the Deflate method varying from minimal compression/fast method (1) to maximal compression/slower method (9); 11-19 = compression via the Enhanced Deflate method varying from minimal compression/fast method (11) to maximal compression/slower method (19).

*Note:* Some older ZIP utilities cannot read files compressed with Enhanced Deflate compression.

**Delete existing output file before starting:** If checked and the output file exists, it is deleted before the compression operation starts.

**Log files that are processed:** If checked, any files added or updated to the output file are logged.

**Skip locked files:** If checked, causes the action to not include any files that are currently locked or in exclusive use by another process.

**Append to existing file:** If checked, appends to an existing ZIP file without creating a temporary file. If unchecked, creates a temporary ZIP file and replaces the original file when done.

**Store filenames in 8.3 format:** If checked and if any of the selected files have long filenames in their path, those filenames are modified to create short filenames in the output file by removing any spaces and taking the first characters (8 for folders/files and 3 for extensions) of each file component. For example, the filename `my new folder\my long filename.extension` would be changed to `mynewfol\mylongfi.ext`.

**Generate PKZIP 2.04g compatible file:** If checked, when creating ZIP files, the action will create ZIP files that can be read by very old ZIP tools such as PKZIP 2.04g, and will be fully compatible with the original PKWare 2.04g file format specification. This has some limitations: filenames in the ZIP file will be converted to the OEM character set first (preventing special characters such as the Euro character from being recorded as part the filename in the ZIP file) and limits the ZIP file to 65,535 files and 4 GB in size. A Compression Factor less than 10 must also be used for full compatibility. When creating gzip files, checking this option will cause long filenames to be ignored.

**Comment:** Specifies a comment to be stored in the output file (optional).

## 3.3 Embarcadero/Borland

### 3.3.1 Make Delphi / C++Builder / RAD Studio

The Make Delphi / C++Builder / RAD Studio custom actions create a step to build Embarcadero / Borland [Delphi](#), C++Builder, RAD Studio, C#Builder, and Turbo projects and groups. It has been tested with *Delphi*, *C++Builder*, *RAD Studio*, *Turbo*, and *Borland Developer Studio* product versions 5 thru 17 (10 "Seattle") and may also work with other versions.

This action processes individual project or multi-project group files. When building `.dpr` and `.bpg` files, it invokes the command-line compiler (`dcc32.exe`) for each project; for `.bdsproj`, `.bdsgroup` files (Delphi 8, 2005, and 2006), it invokes the Delphi/BDS command-line compiler (`bds.exe`) for the project or group specified, and for `.dproj`, `.cbproj`, and `.groupproj` (Delphi 2007 and later) files, it invokes MSBuild (`msbuild.exe`) for the project or group specified. All build output from the compiler is logged. If any errors occur while building, the error output is logged and the action fails.

#### Notes:

- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- See the Embarcadero sample for example of the Make Delphi and Make C++Builder actions.
- Use the Make Delphi Prism action for building Delphi Prism projects and solutions.
- This action uses the highest installed version of Delphi it can find. To call another version when multiple versions are installed, specify the `rsvars.bat` path and/or Delphi/MSBuild executable in the override fields on the Options tab.

#### Project/Group Tab

#### Versions Tab

#### Properties Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

#### Version/Property Handling

For each project that is processed, if any values are specified on the Versions or Properties tabs, the following logic is performed before building the project:

#### Project files:

- Versions/properties are added/updated in the project file.

#### Win32 Delphi projects:

- If the Update DOF checkbox is checked, the `.dof` file is updated with any settings from the step, the Include Version Info setting is enabled, and the Auto-Increment flag is updated to match the step settings (enabled if the step is set to increment versions). This step applies only for Delphi 7 and earlier.
- If the project file does not contain a directive to include its `.res` file (`{ $R *.res }` or `{ $R ProjectFile.res }`), the reference is added.

**Projects that reference or contain a ProjectFile.res file:**

- A custom RC file named <ProjectFile>.vrc in the project directory with a VERSIONINFO resource is created/updated if it differs from the values specified in the step.
- If the associated <ProjectFile>.vres file doesn't exist or is older than the .vrc file, <ProjectFile>.vrc is compiled to <ProjectFile>.vres with the Borland resource compiler (BRCC32.exe).
- If ProjectFile.res does not exist, does not contain a VERSIONINFO resource, or the VERSIONINFO resource information is not up-to-date, the file is created/updated with the VERSIONINFO resource from <ProjectFile>.vres.

**.NET projects:**

- Versions/properties are added/updated in the project's AssemblyInfo file and/or .dpr/.dpk file.

These steps ensure that the compiled executable contains the version information specified by the user, since the Delphi command-line compiler ignores any version property settings made in the IDE, and will also result in subsequent builds in the IDE matching the version/property settings configured in the Make Delphi step.

**Troubleshooting**

File not found errors (i.e.: Sample.dpr(8) Fatal: File not found: 'abc.dcu', and other errors relating to missing files) can occur when building a Delphi project or group even though the same projects compile correctly within the IDE. This is not due to issues with Visual Build, but rather with the Borland Delphi command-line compiler behavior. When building projects from the IDE, the directories defined in Environment Options are used. However when the command-line compiler (dcc32.exe) is invoked, these directory definitions are not used, and must be defined elsewhere. See the following entries in the Delphi Help file for additional information: Dcc32.exe, Dcc32.cfg, Compiler Directives, Directory Options.

**3.3.1.1 Project/Group Tab**

This tab of the Make Delphi action configures information about the Delphi project or group to be built.

**Filename:** The Delphi, C++Builder, C#Builder, or Turbo project, package or group file to build. Individual .bdspproj, .dpr, .dpk, .dpkw, .dproj, .cbproj, .proj project files and .bdsgroup, .groupproj, and .bpg project group files are supported.

**Don't build:** Does not build any projects, but updates other settings such as versions and project properties.

**Update project .res file:** If checked, updates the project .res file with any property or version changes (applies only when *Don't build* is checked -- when building, the .res file will also be updated with version and property changes). If unchecked, only the property and version information in source files (project file, .rc, .dof) will be updated when not building.

**Force build:** Forces a build of all projects (applies only to Delphi/BDS 8.0 and greater) rather than a make.

**Continue building projects on build failure:** Continue building other projects if a project in a group fails to build (applies only for Delphi 7 earlier).

**Log a list of failed projects when done building:** If any projects fail to build and the continue build option is checked, each failed project is logged at the end of the custom action (applies only to project groups for Delphi 7 and earlier).

**Update DOF file with any property or version changes:** If checked, in addition to updating a

custom `.rc` file's VERSIONINFO resource with any version/property changes, the `.dof` file is also updated with any changes, and any properties/versions not specified by the action are read from the `.dof` file and updated in the custom `.rc` file.

**Restore read-only attribute of any files modified by action:** If checked, any project files modified for the build (to set/increment versions, properties, etc.) which are marked read-only before modification will be set back to read-only after being modified.

**Configuration to build:** The configuration to build (applies only to Delphi 2007 and later when building with MSBuild).

**Platform to build:** The platform to build (applies only to Delphi 2007 and later when building with MSBuild).

### 3.3.1.2 Versions Tab

This tab of the Make Delphi action configures what version settings to apply to the projects being processed. It can set or increment project versions in the project file, the resource file's VERSIONINFO resource, DOF file (Win32 Delphi projects), and .NET attributes in the `.dpr/.dprk` files or AssemblyInfo file to values that you specify. The assembly version checkbox applies only for .NET projects.

**Increment Version:** The increment version option will increment the project version. This ensures that your installs work correctly and copy updated files when comparing the file version. The FileVersion and/or ProductVersion fields can be incremented. Normally, the Build number (last number) is incremented, but the Revision number (3rd number) can be incremented instead by checking that checkbox.

**Set Version:** The set version option will set the project version to the specified value. This is useful if you want your executables to be marked with a specific build number. The version should be entered in the format 9.9.9.9 (e.g., 1.0.1.12). The version in the project's RC file(s) is updated before building. The raw version will be stored in the required "9, 9, 9, 9" format, while the string version will be stored exactly as entered.

When using the Increment or Set Version radio button, either one or both of the File or Product version checkboxes should be checked to indicate which version(s) to increment or set.

*Note:* To set different values for the version fields, create multiple Make Delphi steps for a single project/group, with the first having its *Don't build* field checked; then set the file version field in one step and the product version (and assembly version if desired) in other steps. These steps could be placed in a subroutine and called multiple times if this functionality was needed for multiple projects or groups.

### 3.3.1.3 Properties Tab

This tab of the Make Delphi action configures setting of properties for the Delphi project or group to be built.

Used to set project properties in the project file, the resource file's VERSIONINFO resource, DOF file (Win32 projects), and .NET attributes in the `.dpr/.dprk` files or AssemblyInfo to values that you specify. This can be useful for quickly updating all projects in a group or project. The following properties can be set: Comments, Company Name, File Description, Legal Copyright, Legal Trademarks, Product Name, Private Build, and Special Build. Any or all the properties can be updated or cleared. The files are only modified if the existing values differ from those specified.

### 3.3.1.4 Options Tab

This tab of the Make Delphi action configures additional options about the Delphi project or group being built.



**Product version to build with:** If blank, the action will locate and call the highest installed version of Delphi / C++Builder / RAD Studio. Otherwise, the action will use the version or path specified in this field. Supported product version values are 3 thru 8, 2005 thru 2010, XE thru XE8, and 10. The drop-down will be populated with the installed versions that were found.

**Do not run rsvars.bat before MSBuild:** If unchecked, the action will call `rsvars.bat` (the equivalent of running from a RAD Studio Command Prompt) before invoking `msbuild.exe` (applies only when building with MSBuild). The version of `rsvars.bat` called is determined by the version or path specified in the previous field.

**Additional command-line options:** This action invokes the `DCC32.exe` (for `.dpr` and `.dpk` files), `BDS.exe` (for `.bdsproj` and `.bdsgroup` files), or `msbuild.exe` (all other project file types in Delphi/C++Builder/RAD Studio 2007+) command-line program to perform builds, and this option can be used to specify additional flags to pass, overriding any `.cfg` file settings. The values entered in this field are passed through directly to the command-line program.

**Command-line compiler to call:** If this field is blank, the action will automatically locate the `dcc32.exe`, `bds.exe`, or `msbuild.exe` compiler to perform the build. For Delphi 2007 and later (`.dproj`, `.cbproj`, `.groupproj` files), this action locates and calls `msbuild.exe` (enter or select `bds` to force the project or group to be built with the IDE [`bds.exe`] rather than `msbuild.exe`); for `.bdsproj` and `.bdsgroup` files, it invokes the associated `bds.exe` compiler; and for `.dpr`, `.dpk`, and `.bpg` files, it invokes the associated `dcc32.exe` compiler.

To explicitly specify the compiler executable to use, enter a full path and filename for `dcc32.exe`, `bds.exe`, or `msbuild.exe` in this field to use that executable rather than the default.

*Note:* To call the 64-bit version of `msbuild.exe` from the 32-bit edition of Visual Build, enter the full path and filename of `msbuild.exe` in the desired Framework64 folder (i.e., `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe`). To call the 32-bit version of `msbuild.exe` from the 64-bit edition of Visual Build, enter the full path and filename of `msbuild.exe` in the desired Framework folder (i.e., `C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe`).

### 3.3.2 Make JBuilder

The Make JBuilder action creates a step to build [Borland/Embarcadero/CodeGear JBuilder](#) projects and groups.

This action processes single project or multi-project group files and invokes the command-line compiler for each project. All build output from the compiler is logged. If any errors occur while building, the error output is logged to the Output pane, and the action aborts.

The action has been tested with JBuilder 7, 8, 9, X, 2005, and 2006 and may also work with other versions. *Notes:* JBuilder must be installed on the build box, and this action requires the Developer or Enterprise editions of JBuilder, since command-line builds are limited to those editions.

**Filename:** The JBuilder project or group file to build. Individual JPX and JPR as well as BPGR and JPGR (project group) files are supported.

**Project target(s) to build:** Specifies the target(s) in the project to build (optional). The default is `make`. Other common targets are `rebuild` and `clean`.

**Continue building projects on build failure:** Continue building other projects if a project in a group fails to build.

**Log a list of failed projects when done building:** If any projects fail to build and the continue build option is checked, each failed project is logged at the end of the custom action (applies only to project groups).

**Additional command-line options:** This action invokes the `jbuilder.exe` command-line program to perform the builds, and this option can be used to specify additional flags to pass. The values entered in this field are passed through directly to the command-line program.

**Override default compiler location:** Normally, the command-line compiler is found by finding the executable associated with the project/group filename extension, extracting its path, and adding `jbuilder.exe` to the path. By providing a full path and filename for this field, the command-line executable specified will be used rather than looking it up. This can be used to specify which compiler to use when more than one version of JBuilder is installed.

**Log the compiler command-line:** Shows the command-line details for each project that is built. This can be useful for debugging purposes.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.3.3 StarTeam

Visual Build provides the StarTeam action to automate access to [Borland](#) StarTeam. The StarTeam action creates a step for configuring server commands. See `StarTeam.bld` for sample usage.

When the step is built, the action invokes the StarTeam `stcmd.exe` command-line executable to perform the requested command. Some options do not apply to all commands.

#### Server Tab

#### Command Tab

#### Global Options Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with StarTeam versions 2005 through 2009 and 12 through 14.3 and may work with other versions as well.

*Notes:*

- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- See the `StarTeam.bld` sample for a project utilizing this action.
- See the `ContinuousIntegration.bld` sample for a sample of incorporating StarTeam into continuous integration builds.

### 3.3.3.1 Server Tab

This tab of the StarTeam action configures information about the server to use.

**Host:** The server to access (optional, blank for local computer).

**Port:** The port to connect on (required).

**Username:** The StarTeam username to login with (optional). If empty, the current the username of the currently logged in Windows user will be used.

**Password:** Password of the StarTeam user (optional).

**Pwd file:** Filename containing the user's password (optional). If specified, it overrides a password specified above.

**Encrypt password file:** Indicates that the password file is encrypted.

**Project:** The project name (required for most commands).

**View:** A view hierarchy (optional). Use the colon (:) as a delimiter between view names. The view hierarchy should always include the root view. For example, "StarDraw:Release 4:Service Packs" indicates that the view to be used is the Service Packs view, which is a child of the Release 4 view and a grandchild of the StarDraw root view. If the view name is omitted, the root (default) view is used. If the view is the only view in that project with that name, you can use only the view name. (This is not recommended because another view with that name could be created at a later date and cause confusion.) The view name in the example is StarDraw. Because this is the root view of the StarDraw project, it could have been omitted.

**Folder:** A folder hierarchy to identify the folder. Use the forward slash (/) as a delimiter between folder names. The folder hierarchy never includes the root folder. Omit the folder hierarchy if the file is in the view's root folder. For example, if the root folder of the view is StarDraw and the hierarchy to your files is "StarDraw/SourceCode/Client", use only "SourceCode/Client".

**Compress data:** Compresses all the data sent between the workstation and the server and decompresses it when it arrives. Without this option, no compression takes place.

**Encryption:** Encrypts all the data sent between the workstation and the server and unencrypts it when it arrives. Without this option, no encryption takes place.

Encryption protects files and other project information from being read by unauthorized parties over unsecured network lines. The encryption types are ordered from fastest to slowest. Each of the slower encryption types is safer than the one preceding it.

### 3.3.3.2 Command Tab

This tab of the StarTeam action configures flags that apply to the command.

**Command:** The StarTeam command to perform (required).

**Files or masks:** Specifies the files to be used in the command by name or by file name-pattern specification (optional, one per line). All options are interpreted using the semantic conventions of UNIX instead of Windows because UNIX's conventions are more specific. This means that "\*", rather than "\*. \*" means "all files." The pattern "\*. \*" means "all files with file name extensions." For example, "star\*. \*" finds `starteam.doc` and `starteam.cpp` but not `starteam`. To find all of these, you could use "star\*". When not specified, the default is "\*". If you use \* rather than "\*" to indicate all files, a UNIX shell expands it into a series of items and passes this series as a group of options to the stcmd command. This can cause problems, for example, when you are checking out

missing files, so it is best to use "\*" to avoid unwanted complications.

Several special characters can be used in file specifications:

- \* Matches any string including the empty string. For example, "\*" matches any file name, with or without an extension. "xyz\*" will match "xyz" and "xyz.cpp" and "xyzutyfj".
- ? Matches any single character. For example, "a?c" will match "abc" but NOT "ac"
- [...] Matches any one of the characters enclosed by the left and right brackets. A pair of characters separated by a hyphen (-) specifies a range of characters to be matched. If the first character following the right bracket ( ] ) is an exclamation point ( ! ) or a caret ( ^ ), the rest of the characters are not matched. Any character not enclosed in the brackets is matched. For example, "x[a-d]y" matches "xby" but not "xey". "x[!a-d]y" matches "xey" but not "xby". A hyphen (-) or right bracket ( ] ) may be matched by including it as the first or last character in the bracketed set. To use an asterisk (\*), question mark (?), or left bracket ( [ ) in a pattern you must precede it with the escape character (which is the backslash \).

**Apply command recursively:** If checked, applies the command recursively to all child folders. If unchecked, the command applies only to the specified folder.

**Working directory for local files:** Overrides the specified folder's working folder or working directory (optional). This is equivalent to setting an alternate working path for the folder.

**Revision to operate on:** Specifies the version to operate on (optional). If blank, the tip version is used. If Nnumber is entered, the specified revision number is used. If Ddate/time is entered, it specifies the as-of date/time used to identify the revision. If Llabel is entered, it specifies the revision or view label to use.

**Configure view:** Configures the view (optional). If blank, the current configuration is used. If Ddate/time is entered, the view is configured using the specified as-of date/time. If Llabel is entered, the view is configured using the specified label. If Pstate is entered, the view is configured using the specified promotion state.

**Comment, description, or name:** The comment, description, or name to apply to the command (optional).

### 3.3.3.3 Global Options Tab

This tab of the StarTeam action configures global options.

**Quiet mode:** Suppresses progress reporting. Without this option, messages about each action appear on the screen as the action is performed.

**Batch mode:** Switches to batch mode. If checked, no interactive error messages are displayed.

**No logo:** Suppresses the copyright notice.

**Halt execution at first error:** Halts execution of the command-line when the first error is encountered. Without this option, execution continues despite errors.

**Case-sensitive folder names:** When the command maps the folder specified on the Server tab to the underlying folder, this checkbox causes the command to differentiate folders based on the case-sensitive spelling of their names. For example, with this option checked, folders named doc and Doc are recognized as different folders. With this option unchecked, either of these folders could be recognized as the "doc" folder.

*Note:* This option has nothing to do with case-sensitivity of file names.

**Apply command to all files needing checkin:** Self-explanatory.

**Do not move labels if already attached:** Self-explanatory.

**Force check in/out:** Self-explanatory.

**Mark change request fixed, requirement complete, or task finished:** Self-explanatory.

**End-of-line conversion:** Determines whether end-of-line conversion is performed for text files.

**File status filter:** Specifies the file statuses that the command applies to.

#### 3.3.3.4 Options Tab

This tab of the StarTeam action configures additional options.

**File attributes:** Specifies changes to file attributes (none, set read-only or make writeable).

**Locking options:** Specifies file locking options (none, lock exclusively, lock non-exclusively, or unlock).

**Success codes:** Specifies exit codes to treat as successful. Possible exit codes are:

- 0 success
- 1 for failure
- 101 if at least one of the specified file patterns did not match
- 102 if none of the specified file patterns matched

For the diff command with the -e option in the following field, the exit codes can be:

- 0 if all the compared files are equivalent
- 1 if an error condition occurs
- 2 if at least one file is different

**Additional options:** Use this field to enter any additional stcmd.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the *StarTeam User's Guide [Using the stcmd Command-line Interface]* section) or *StarTeam Command-line Tools Help* for details on the available command-line flags).

**Location of StarTeam executable:** If not specified, the action locates stcmd.exe automatically. If the action is unable to locate the stcmd.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to stcmd.exe here.

## 3.4 Files

### 3.4.1 Burn CD/DVD

The Burn CD/DVD action creates a step to burn files or an ISO image to a writeable or rewriteable CD or DVD drive or to create an ISO image. It supports processing of subdirectories, CD-R, CD-RW, DVD-R, DVD+R, DVD-RW, DVD+RW, DVD-RDL, DVD-RAM, HD-DVD, HD-DVD-RAM, Blu-ray, and Blu-ray E drives and media, supports processing of subdirectories, include and exclude of files and folders, importing previous sessions, configurable logging of operations, a test mode to calculate the size of the image without burning, verifying, and more. When built, the action finds all matching files and folders (recursively searching for matches if specified), and burns them to the recordable drive or ISO image.

When the step completes, the following temporary macros are created or updated:

- **BURN\_FILE\_COUNT** = The number of files and folders that were burned to the disc or image
- **BURN\_TOTAL\_SIZE** = The size (in MB) of the burned image

## Burn Tab

### Options Tab

### Attributes Tab

*Notes:*

- See the Files.bld sample for a project utilizing this action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.4.1.1 Burn Tab

This tab of the Burn CD/DVD action specifies the operation to perform and the source and destination files.

**Operation:** The operation to perform: Burn files to disc, Burn ISO image to disc, or Create ISO image.

**Source folder:** The source location for files to be burned (required; does not apply when creating ISO images).

*Note:* You can also specify a path and filename or path and file mask in this field (instead of using the Include field).

**Files to burn:** Used to specify files or folders to specifically include or exclude (optional).

**Include subdirectories:** Determines if subdirectories will be searched recursively for burning.

**Exclude empty subdirectories:** If checked, empty, matching subdirectories will not be included when burning (applies only when *Include subdirectories* is checked).

**Burner drive:** The drive letter of the destination burning device. Click Properties to view information about the current drive and media; click Refresh to refresh the drop-down list with all available burning devices.

**ISO filename:** The ISO filename to burn when burning an ISO to disc or to create when creating an ISO image (required).

**Volume label:** A label to apply to the burned disc or ISO image (optional, does not apply when burning an ISO image).

**Output path:** The root directory to create *on the disc or image*, in which all burned files will be written to (optional).

#### 3.4.1.2 Options Tab

This tab of the Burn CD/DVD action specifies additional options about the burn activity.

**Calculate image size only (no burn):** If checked, no files will be burned, but any enabled logging options will be performed. Useful for debugging purposes.

**Test burn:** If checked, the burn device will be placed in test mode instead of actually burning the image.

**Erase disc first:** For rewriteable discs, this will erase the disc before burning (applies only when using

rewriteable media and drive).

**Quick erase:** If checked, performs a quick erase, otherwise a full erase is performed (can take much longer).

**Import session:** If checked, imports a previously burned session into the new session. To import previous sessions, use blank or 0 in the following field; to import a specific session by number, enter that number in the following field. The Folder can be used to specify where the previous session(s) will be imported into the new session (use blank or \ to import to the root of the new session).

**Max burn speed:** Used to specify a maximum burn speed in KB/sec (optional). Leave blank to burn at the max supported by the drive/media.

**Cache size:** Specifies the buffer size to use when burning in MB (optional). Leave blank to use the default cache size.

**Buffer underrun protection:** Helps to protect from buffer underruns when burning.

**Perform OPC:** If checked, monitors and maintains the quality of disc writing (also called Dynamic Power Control or Direct Read During Write).

**Joliet file system:** Causes the system to create Joliet file system on the burned disc or image. Select this option if you want to use file names that contain up to 64 characters in length, including spaces. Joliet also records the associated DOS-standard name (8+3 characters) for each file so that the disc may be read on DOS systems or earlier versions of Windows.

**UDF Bridge support:** If checked, the disc/image will be formatted in compliance with OSTA UDF version 1.50.

**Create a bootable disc/Boot image:** Creates a bootable image using the specified file.

**Log every x% completed:** Specifies a percentage to log the progress of the burn (optional or 0 for no progress logging).

**Log files and folders that are processed:** If checked, all folders and files add to the image will be logged when the step is built.

**Finalize disc:** Used to finalize/close the media in the device after burning . If finalized, no more data can be written to the media, even if there is free space available on it.

**Eject disc after burn:** Ejects the disc after the burn is complete.

### 3.4.2 Copy Files

The Copy Files action creates a step to copy, move, delete, and synchronize files and folders. It supports processing of subdirectories, incremental copying (copying only files that have changed since previous copy), synchronization of folders (by purging extra files from the destination folder), including and excluding files and folders via file masks, moving of files (delete after copy), configurable logging of operations, a debug mode to only display the files that would be processed, and more.

When built, the action finds all matching files and folders (recursively searching for matches if specified), copies any matching files (creating destination folders as necessary), deletes the source files after successfully copying if moving, and also deletes any files/folders in the destination that do not exist in the source folder if the purging.

When the step completes, the following temporary macros are created or updated:

- **COPYFILES\_COPY\_COUNT** = The number of files that were copied

- **COPYFILES\_DELETE\_COUNT** = The number of files deleted by the step
- **COPYFILES\_SKIP\_COUNT** = The number of files skipped by the step

### Copy Tab

### Options Tab

### Errors Tab

### Attributes Tab

### Robocopy Tab

### Advanced Tab

### Remote Tab

*Note:* See the Files and Server samples for examples of the Copy Files action.

#### 3.4.2.1 Copy Tab

This tab of the Copy Files action specifies what should be copied, moved or deleted.

**Source folder:** The source directory to copy files from (optional). This can be empty if the *Ignore non-existent or blank source folder* and *Synchronize* options are checked to delete files and folders from the destination folder.

*Note:* You can also specify a path and filename or path and file mask in this field (instead of using the Include field).

**Dest folder:** The destination folder to copy files to (required) and/or synchronize files from.

**Files to copy:** Used to specify files or folders to include or exclude when copying (optional).

**Copy subdirectories:** Determines if subdirectories will be searched recursively for copying and purging.

**Copy empty subdirectories:** If checked, empty subdirectories in the source will be created in the destination.

**Copy all files to root destination folder (flat copy):** If unchecked (the default), the source folder directory structure will be created in the destination path when copying files. If this option is checked, all files from all source directories will be copied to the root destination folder (no subdirectories will be created). Be careful with this option -- if duplicate filenames exist in different folders in the source, the last-copied duplicate filename will replace all prior copies. It is also recommended that this option not be used with the synchronize and move options.

*Note:* Does not apply with Robocopy.

**Copy only files that have changed (incremental copy):** Enables incremental copying of files. Before copying a file, if this option is checked, the action compares the size and timestamp of the source and destination files and only copies the files if the file does not exist in the destination folder or the size or timestamps are different. Any files that are already up-to-date in the destination will not be copied.



**Skip files from source that are older than destination:** If checked, when a newer file already exists in the destination path, the source file will not be copied (the file sizes are not compared). If unchecked, every matching file that is different in size or timestamp (newer or older) will be copied to the destination.

**Delete destination files and folders that no longer exist (synchronize):** If checked, any files in the destination folder that do not exist in the source folder will be deleted from the destination folder. Use in conjunction with incremental copy to synchronize the contents of two folders. The Include/Exclude fields are also used to include or exclude files from the synchronize operation.

*Note: Be very careful when using this option, as many files can be quickly deleted when this option is specified. It is advisable to use the *Do not copy* option when testing a synchronize or delete operation to show which files would be removed. The Delete Files action can also be used to delete files.*

**Move files:** If checked, after successfully copying each file, the file is deleted from the source folder.

**Move/delete source folders:** If checked, source folders are also deleted if the folder is empty after all matching files are moved.

*Note: To delete files, check the *Ignore non-existent or blank source folder* checkbox on the Options tab, clear the Source Folder field, enter the folder to delete files from in the Dest Folder field, and check the *Delete destination files...* option. Include and exclude options can also be entered to specify which files to delete.*

### 3.4.2.2 Options Tab

This tab of the Copy Files action specifies additional options about the copy activity.

**Do not copy:** If checked, no files will be moved, copied, or synchronized, but any enabled logging will be displayed and a count of files that would have been copied, moved, or synchronized will be logged. Useful for debugging purposes.

**Log files and folders that are copied, moved, or purged:** If checked, any files or folders that are processed will be logged.

**Log files that are already up-to-date:** If the incremental copy option is checked, and this option is checked, files that matched but were not copied because the destination file already matched will also be logged.

**Skip files that already exist in destination:** If checked, when a file already exists in the destination path, the source file will not be copied (regardless of its timestamp or size).

*Note: Does not apply with Robocopy.*

**Only copy files that already exist in destination:** If checked, the source file will be copied only if the file already exists in the destination path.

**Ignore non-existent or blank Source folder:** Allows an empty or non-existent folder to be entered in the Source Folder field. Useful for deleting files (empty Source Folder entered) or if the step should succeed even if the Source folder does not exist.

*Note: Be very careful when using this option, as many files can be quickly deleted when this option is checked and the *Delete destination files and folders that no longer exist option* on the Copy tab is also checked. It is advisable to use the *Do not copy* option when testing a synchronize or delete operation to show which files would be removed. The Delete Files action can be used instead to explicitly delete files.*

**Copy files in restartable mode:** If checked, the file copy operation uses the restartable flag to allow Windows to continue a partial copy that is aborted.

**Copy using short filenames:** If checked, uses the short (8.3) file or folder name for each file that is copied.

**Additional destination paths:** Specifies additional destination folders to copy files to and/or synchronize files from (optional, one per line).

### 3.4.2.3 Errors Tab

This tab of the Copy Files action specifies options regarding error handling.

**Wait for share names to be defined:** If checked, retries copying of a file if the destination folder references a network share that does not exist. If unchecked, retries are only performed for network and sharing violation errors.

**Continue processing remaining files on failure:** If checked and an error occurs while processing, the action will continue processing the remaining files (but still fail the step when done). If unchecked, the step will fail as soon as an error occurs.

*Note:* Does not apply with Robocopy.

**Fail the step if no matching files found:** If checked and no matching files are found to copy, the step will fail. If unchecked, the step will succeed even if no matching files were found.

**Fail the step if unable to purge files or folders:** If checked and the purge option is also checked, if any purged files are folders cannot be deleted, the action will fail.

*Note:* Does not apply with Robocopy.

**Fail the step if error occurs retrieving file information:** If checked and the incremental is checked, if retrieving file information (size or timestamp) for any file fails, the step will fail. If unchecked and an error occurs retrieving file information, the files will be considered different and will be copied.

*Note:* Does not apply with Robocopy.

**Number of retries on failed copies:** Number of times to retry copying a file (required). Can be 0 to disable retry.

**Seconds to wait between retries:** Time to wait between retries (required).

### 3.4.2.4 Attributes Tab

This tab of the Copy Files action specifies attributes of files to match for copying.

**Maximum modification date:** If a date/time or number of days is specified, only files with a modification date or age less than or equal to that date/age will be copied.

**Minimum modification date:** If a date/time or number of days is specified, only files with a modification date or age greater than or equal to that date/age will be copied.

**Maximum access date:** If a date/time or number of days is specified, only files with an access date or age less than or equal to that date/age will be copied.

**Minimum access date:** If a date/time or number days of is specified, only files with an access date or age greater than or equal to that date/age will be copied.

**Maximum size:** If a size is specified, only files with a size less than or equal to that size (in bytes) will be copied.

**Minimum size:** If a size is specified, only files with a size greater than or equal to that size (in bytes) will be copied.

**Attributes:** Specifies files and folders to include or exclude based on their attributes (only the hidden attribute applies for folders).

**Clear on source:** If checked, after copying a file with the archive attribute set, the action clears the archive attribute on the source file.

**Keep on copy:** If checked, read-only files that are copied will retain their read-only attribute in the destination file. If unchecked, the read-only attribute will be removed from the destination file after copying.

*Note:* The *Exclude non-* options do not apply with Robocopy.

### 3.4.2.5 Robocopy Tab

This tab of the Copy Files action specifies options for calling Robocopy.

**Use Robocopy:** If checked, the action will call Robocopy to copy files. The other options on this tab and the Advanced and Remote tabs apply only when this option is checked.

**Copy using unbuffered I/O:** If checked, copies using unbuffered I/O (recommended for large files). Requires Robocopy v6 or later (installed with Windows 8 and later).

**Do multi-threaded copies:** Specifies the number of threads for multi-threaded copying (optional, between 1 and 128). When specified, the use of a log file is recommended for better performance.

**Log file:** Specifies a file to log output to (optional).

**Append to existing log file:** If checked, appends to the log file; if unchecked, overwrites any existing log file.

**Unicode log file:** Outputs to Unicode log file.

**Log to build output and log file:** If checked, logs output to the specified log file, Output pane, and Visual Build log file.

**Copy all file info:** If checked, copies all file info (data, attributes, timestamps, security, owner info, and auditing info).

**Copy no file info:** If checked, copies no file info.

**Don't log header info:** Do not log job header.

**Don't log summary info:** Do not log job summary.

**Additional Robocopy options:** Specifies additional arguments to Robocopy.

**Override Robocopy executable filename:** Specifies the Robocopy executable filename (optional). If not specified, robocopy.exe must exist in the PATH environment variable.

### 3.4.3 Create Folder

The Create Folder action can be used to create a folder on a local drive or network path.

**Folder:** The drive+path or network path to create (required). The action creates all subdirectories that don't exist.

**Fail the step if the folder already exists:** If checked and the folder exists, the step will fail.

*Note:* See the Recurse sample for an example of the Create Folder action.

### 3.4.4 Delete Files

The Delete Files action creates a step to delete files and folders. It supports processing of subdirectories, including and excluding files and folders via file masks, configurable logging of operations, a debug mode to only display the files that would be deleted, and more.

*Note:* Be very careful when using this action, as many files can be quickly deleted. It is advisable to use the *Do not delete* option when testing to show which files would be deleted.

When the step completes, the following temporary macros are created or updated:

- **DELETEFILES\_COUNT** = The number of files that were deleted

**Folder:** The directory to delete files from (required).

*Note:* You can also specify a path and filename or path and file mask in this field (instead of using the Include field).

**Files to delete:** Used to specify files or folders to specifically include or exclude when deleting (optional).

**Ensure file is writeable before deleting:** Determines if read-only files will be deleted.

**Search subdirectories:** Determines if subdirectories will be searched recursively for deleting.

**Delete empty folders:** If checked, empty subdirectories as a result of deleting files will also be deleted.

**Don't delete root folder:** If checked, the root folder will not be deleted even if all files and folders in it are deleted.

**Continue processing remaining files:** If checked, and a matching file can't be deleted, processing of the remaining files will continue (the step will still report failure on completion).

**Do not delete:** If checked, no files will be deleted, but any enabled logging will be displayed and a count of files that would have been deleted will be logged. Useful for debugging purposes.

**Log files that are deleted:** If checked, any files that are deleted will be logged.

#### Attributes Tab

*Note:* See the Recurse sample for an example of the Delete Files action.

### 3.4.5 Delete Folder

The Delete Folder action can be used to delete a folder or network path and all files within it. This action uses the `cmd.exe RMDIR` command to delete a directory.

**Folder to delete:** The drive+path or network path to delete (required).

**Delete subfolders:** If checked, all subfolders and files are recursively deleted as well. If unchecked and the folder contains subfolders, the action will fail and the folder will not be deleted.

**Log command-line:** Logs the command-line used to delete the folder.

*Notes:*

- Be very careful when using this action, as many files and folders can be quickly and permanently deleted.
- See the Recurse sample for an example of the Delete Folder action.

### 3.4.6 File Checksum

The File Checksum action creates a step to calculate the checksum for a file.

For the Add action, the following temporary macro will be created when the step is built:

**FILE\_CHECKSUM\_VALUE:** The computed checksum/hash value for the file.

#### Checksum Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

##### 3.4.6.1 Checksum Tab

This tab of the File Checksum action specifies the information about the file to be processed.

**Action:** The action to perform.

- *Compute:* Calculates the checksum/hash for the specified filename. If the *Database filename* field is blank, the checksum value will be stored in the FILE\_CHECKSUM\_VALUE temporary macro, otherwise, the checksum value will be stored in the XML database file.
- *List:* Lists all files in the XML database.
- *Verify:* Verifies all entries in the XML database.

**Filename:** The filename to compute the checksum for (required for Add action).

**Hash type:** The type of checksum to calculate (MD5, SHA, or Both).

**Database filename:** The XML database file to store or read hash values to/from.

**Do not store full path name:** If checked, the full path name is not stored in the XML database file.

**Remove base path from entries:** If checked, the base path will be removed from each entry in the XML database file.

##### 3.4.6.2 Options Tab

This tab of the File Checksum action configures additional options.

**Additional command-line options:** Use this field to enter any additional command-line flags to be

passed to `fciv.exe` (optional).

**Override FCIV executable filename:** Use this field to specify the `fciv.exe` filename (optional). If not specified, `fciv.exe` in the `Tools` directory in the VisBuildPro install path will be used if *Full installation* is selected during installation. Otherwise, the executable must be in the `PATH` environment variable.

### 3.4.7 List Files

The List Files action creates a step to log the timestamp, size, and file version (if available) for files in a folder and subfolders that match a given set of criteria.

The following temporary macros will be created when the step is built:

- **LISTFILES\_COUNT:** The number of files that were found to match.
- **LISTFILES\_SIZE:** The total size of all matching files.

#### Files Tab

#### Attributes Tab

*Note:* See the VStudio and VisBuildPro samples for examples of the List Files action.

#### 3.4.7.1 Files Tab

This tab of the List Files action specifies the file to be listed and other output options.

**Folder:** The folder to search for files in.

*Note:* You can also specify a path and filename or path and file mask in this field (instead of using the Include field).

**Files to list:** Used to specify files or folders to include or exclude (optional).

**Search subdirectories (recursive):** If checked, any subfolders of the main folder are also searched for matching files.

**Log output:** If checked, the output is written to the build output.

**Output filename:** If a filename is specified, the output is written to the specified file.

**Ensure file is writeable:** If checked and the output file already exists with read-only attributes, the read-only attribute is removed before overwriting.

**Append to existing file:** If checked, the file will be appended to if it already exists. If unchecked, the file will be replaced if it exists.

**Write BOM:** If checked, the action writes a byte order mark to the beginning of the file. Applies only when creating or replacing a file and using an encoding of UTF-8 or Unicode.

**Encoding (code page):** The encoding or Windows code page that will be used when reading and writing the file (optional). If left blank, the system code page (`CP_ACP`) will be used or the encoding specified by the BOM if the file contains a BOM (byte order mark). Other common encodings for files are UTF-8, Unicode (UTF-16 LE), Windows-1252, Cyrillic, Chinese, and other encodings or code pages supported by Windows (see the [Microsoft Global Development](#) web site and [MSDN Unicode topics](#) for more details).

**List empty folders:** If checked, empty folders will also be listed.

### 3.4.7.2 Attributes Tab

This tab of the Burn CD/DVD, Delete Files, List Files, Process Files, Rename Files, Set File Attributes, Enhanced Unzip Files, and Enhanced Zip Files actions specifies attributes of files to include or exclude.

**Maximum modification date:** If a date/time or number of days is specified, only files with a modification date or age less than or equal to that date/age will be listed.

**Minimum modification date:** If a date/time or number of days is specified, only files with a modification date or age greater than or equal to that date/age will be listed.

**Maximum access date:** If a date/time or number of days is specified, only files with an access date or age less than or equal to that date/age will be listed.

**Minimum access date:** If a date/time or number days of is specified, only files with an access date or age greater than or equal to that date/age will be listed.

**Maximum size:** If a non-zero size is specified, only files with a size less than or equal to that size (in bytes) will be listed.

**Minimum size:** If a non-zero size is specified, only files with a size greater than or equal to that size (in bytes) will be listed.

**Attributes:** Specifies files and folders to include or exclude based on their attributes (only the hidden attribute applies for folders).

### 3.4.8 Process Files

The Process Files action creates a step to process multiple files in a folder (and optionally in subfolders) that match a given file mask. This action generates a list of all files matching the specified criteria, and then performs all child steps once for each file that matches. Create one or more child steps of any type (including Subroutine Call actions) and use the macros defined below to operate on each matching file.

**Folder:** The folder to search for files in.

**Recursively search sub folders:** If checked, any sub folders of the main folder are also searched for files.

**Process once for each folder:** If this option is checked, rather than once for each file, the child steps are invoked once for each folder that contains one or more matching files.

**Process empty folders:** If checked, even empty folders will be matched/processed (enabled only if *Process once for each folder* is checked).

**Fail step if no matching files found:** If checked and no matching files are found, the step will fail instead of being skipped when building.

**Log matches found while searching:** If checked, each matching file is logged while scanning for matching files (filenames are also logged for each iteration when child steps are being processed even when this option is unchecked).

**Files to process:** Specifies the files and folders to include or exclude.

#### Attributes Tab

Each child step is built once for each matching file, and the following system macros are defined for each matching file:

- **PROCFILES\_FULLPATH:** The full path and filename of the matching file (system macro).
- **PROCFILES\_FILENAME:** The base filename without path.
- **PROCFILES\_FILE\_DIR:** The relative directory (if any) from the root folder.
- **PROCFILES\_ROOT\_DIR:** The root directory that was specified in the Folder field.
- **PROCFILES\_COUNT:** The number of matching files for the last Process Files action.
- **PROCFILES\_CURRENT:** The index of the current file being processed (1-based).

The above macros can be used in child steps to process each file that matches the search.

*Notes:*

- See the Chain, Recurse, and Script samples for examples of the Process Files action.
- This is an iterative action which repeats once for each matching file. If used with the Rebuild Selected build action or Test button, the matching files will be logged, but the child steps will not be processed once for each matching file; see the Methods of Building help topic for other ways to test iterating steps.
- A Process Files action must be a child of a subroutine step (it can't be the subroutine entry point). Use a Group action as the subroutine entry point and the Process Files step as a child of it.

### 3.4.9 Read File

This action creates a step to read or filter text from a file, macro, or string and store in a temporary macro. The Loop action can be used with this action to process multiple lines or values.

#### File Tab

#### Text Tab

##### 3.4.9.1 File Tab

This tab of the Read File action specifies the text to read and regular expression matching properties.

**Read text from:** The file or string to read text from (required).

**Case-sensitive matching:** If unchecked, matches are case-insensitive; if checked the case must match when searching.

**Dot does not match newline:** Causes . (period character) to not match newline characters (negates Perl /s modifier).

**Allow comments and ignore white space (Perl extended mode):** If checked, white space outside of character classes is (mostly) ignored and anything following an unescaped # character is treated as a comment.

**Encoding (code page):** The encoding or Windows code page that will be used when reading and writing the file (optional). If left blank, the system code page (CP\_ACP) will be used or the encoding specified by the BOM if the file contains a BOM (byte order mark). Other common encodings for files are UTF-8, Unicode (UTF-16 LE), Windows-1252, Cyrillic, Chinese, and other encodings or code pages supported by Windows (see the [Microsoft Global Development](#) web site and [MSDN Unicode topics](#) for more details).

**Convert null bytes to spaces:** If checked, any null bytes in a binary file will be converted to spaces. If unchecked, only data up to the first null byte in a binary file will be read.



### 3.4.9.2 Text Tab

This tab of the Read File action specifies the text to search for in the text.

**Text to find:** The text or regular expression to find (optional). This can be blank to read the entire file/text or specify a regular expression for the text to match. The Loop action can be used to process all matching lines or values.

*Notes:*

- To insert literal bracket [ ] and percent sign % characters within this field, they must be doubled up (just as in any other field), since these are normally interpreted by Visual Build as referencing script code and macros within a field.
- . (dot) matches newline characters by default. Uncheck Dot does not match newline option to prevent . from matching newline characters and perform line-by-line matching.

**Include all matches:** If checked, each match of the expression will be stored in an array element; if unchecked only the first match will be stored.

**Match handling:** Specifies how matching text is stored. If *Store entire match in string* is selected, all matching text will be stored. If *Store matching groups in array* is selected, each captured group will be stored in an array element in the temporary macro.

**Fail if no matching text found:** If checked and the file is empty or nothing matches the expression, the step will fail. If unchecked and no match is found, an empty string will be stored in the temporary macro.

**Log matches:** If checked, all matching text found in the file will be logged when the step is built.

**Macro name:** The name of the temporary macro in which to store the text that was read from the file (optional, defaults to storing in the **READ\_FILE\_VALUE** if not specified).

**Description:** The description of the temporary macro where the text is stored (optional).

**Do not escape special characters:** Does not escape (double) special characters in the file when storing in a macro.

**Append to macro value:** If checked and the macro already exists, the found text will be appended if the macro already exists. If unchecked, any existing macro value will be replaced.

### 3.4.10 Read INI

This action creates a step to read values from an INI file and store in a temporary macro. The Loop action can be used to process multiple matching values.

**Filename:** Name of the INI file to read (required). If no path is specified, the file in the Windows folder (%WINDIR% system macro) will be read.

**Section:** Section of the file to update (optional; retrieves all section names into array if blank).

**Value name:** Value name to create or delete (optional; if blank, retrieves all value names in section into array).

**Default value:** The value to use if it doesn't exist (optional).

**Macro name:** The name of the temporary macro in which to store the value that was read from the file (optional, defaults to storing in the **READ\_INI\_VALUE** if not specified).

**Description:** The description of the temporary macro where the text is stored (optional).

**Do not escape special characters:** Does not escape (double) special characters in the INI file values when storing in a macro.

**Append to macro value:** If checked and the macro already exists, the found text will be appended if the macro already exists. If unchecked, any existing macro value will be replaced.

**Log matches:** If checked, all matching text found in the file will be logged when the step is built.

**Fail step if file, section, or value not found:** If checked and the file, section, or value doesn't exist, the step will fail. If unchecked, the default value will be used.

*Notes:*

- See the Files sample for an example of the Read INI action.
- The READ\_INI system macro can also be used to read values from an INI file.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.4.11 Read XML

This action creates a step to read element or attribute values from an XML file and store in a temporary macro. The Loop action can be used to process multiple matching values.

#### File Tab

#### Namespaces Tab

#### XPath Tab

#### Results Tab

*Notes:*

- This action requires MSXML version 3, 4 or 6 to be installed (the highest available version will be used).
- See the XML.bld sample for sample usage.
- The Write XML action can be used to write values to an XML file.
- The READ\_XML system macro can also be used to read XML files.

#### 3.4.11.1 File Tab

This tab of the Read XML action specifies information about the file to be read.

**Filename:** Name of an existing XML file to update (required).

**Do not validate against DTD or schema:** If checked, the document will not be validated against a document type definition (DTD), schema, or schema cache.

**Do not resolve external definitions:** If checked, external definitions, resolvable namespaces, DTD external subsets, and external entity references will not be resolved at parse time, independent of validation.

**Prohibit inclusion of a DTD:** Determines whether inclusion of a DTD is allowed or prohibited.

**Preserve white space:** If checked, preserves white space in the document.

**Enable XSLT script code execution:** Determines whether <msxsl:script> element functionality is allowed.

**Enable document function:** Determines whether the document function is allowed.

**Use inline schemas for validation:** Determines whether inline schemas should be processed for validation.

### 3.4.11.2 XPath Tab

This tab of the Read XML action specifies the XPath expression to query.

**XPath expression:** An XPath expression that identifies the nodes to query (required).

**Force XPath query language for MSXML v3:** If checked and MSXML v3 is used, the query language will be set to XPath (XPath is always used by newer versions of MSXML).

**Attribute to query:** The name of an attribute within the specified nodes to query (optional). If blank, the text for the first node matching the XPath expression will be used.

**Retrieve all matching nodes:** If checked, all matching nodes will be retrieved; if unchecked, only the first matching node will be retrieved. The Loop action can be used to process all matching nodes and stored as an array in the temporary macro.

**Fail step if element or attribute not found:** If the specified element or attribute does not exist and this option is checked, the step will fail. If it does not exist and this option is unchecked, the default value will be used.

**Default value:** The text to set the matching element or attribute value to (optional).

**Return MSXML object:** If checked, the matching MSXML node list object will be stored in the temporary macro (useful for iterating over a collection of elements or attributes with the Loop action), and the *Attribute to query*, *Retrieve all matching nodes* (all matching nodes will be returned), *Default value* and *Append* options do not apply.

If unchecked, the matching value(s) will be stored in an array value in the specified macro.

*Note:* Bracket characters [ and ] and the percent sign character % within a macro's value have special meaning in a macro value.

### 3.4.11.3 Results Tab

This tab of the Read XML action specifies how the results will be handled.

**Macro name:** The name of the temporary macro in which to store the matching values read from the file (optional, defaults to storing in the **READ\_XML\_VALUE** if not specified).

**Description:** The description of the temporary macro where the text is stored (optional).

**Do not escape special characters:** Does not escape (double) special characters in the XML file when storing in a macro.

**Append to macro value:** If checked and the macro already exists, the found text will be appended if the macro already exists. If unchecked, any existing macro value will be replaced.

### 3.4.12 Rename Files

The Rename Files action creates a step to rename files and folders. When built, the action finds all matching files and/or folders (recursively searching for matches if specified) and renames any matching files or folders based on the search and replace values specified.

When the step completes, the following temporary macros are created or updated:

- **RENAMEFILES\_COUNT** = The number of files and folders that were renamed
- **RENAMEFILES\_PROC\_COUNT** = The number of matching files and folders processed by the step

#### Rename Tab

#### Options Tab

#### Attributes Tab

*Note:* See the Files.bld sample for a project utilizing this action.

#### 3.4.12.1 Rename Tab

This tab of the Rename Files action specifies what files or folders should be renamed.

**Folder:** The source directory to rename files and folders in (required).

*Note:* You can also specify a path and filename or path and file mask in this field (instead of using the Include field).

**Items to process:** Used to specify files and folders to specifically include or exclude (optional).

**Process subdirectories:** Determines if subdirectories will be searched recursively for renaming.

**Rename matching folders:** If checked, matching folders will also be renamed. If unchecked, only matching filenames will be renamed.

**Do not rename matching files:** If checked, only matching folders will be renamed. If unchecked and Rename matching folders is checked, matching folders and files will be renamed.

**Do not rename:** If checked, no files or folders will be renamed, but any enabled logging will be displayed and a count of files and folders that would have been renamed will be logged. Useful for debugging purposes.

**Log files and folders that are renamed:** If checked, any files or folders that are renamed will be logged.

**Log files and folders that are processed but not renamed:** If checked, files or folders that matched the include/exclude list but already matched the replace value will also be logged.

#### 3.4.12.2 Options Tab

This tab of the Rename Files action specifies additional options about the rename activity.

**Replace existing files/folders with names matching new names:** If checked, any existing files or folders matching already the names that any files or folders are renamed will be deleted before first.

**Ensure file to replace is writeable:** Makes any read-only files from the previous option writeable before deleting.

**Perform case-sensitive matching:** Determine whether matching of file and folder names will be case-sensitive.

**Replace first match only:** If checked, only the first match is replaced; if unchecked, all matches are replaced.

**Regular expression to match names with:** A regular expression to match file and folder names on.

**Expression to replace matches with:** A literal string or expression to replace the matching filename or folder name string with.

### 3.4.13 Replace in File

This action creates a step to find and replace text in a text or binary file. It can be used with the Process Files action to replace text in multiple files.

When built, the action reads the input file into memory, performs the find and replace operation, and writes the updated contents with any replacements to the output file. If no matches are found and the append field is not empty, the text from the append field is appended to the file. Otherwise, if no matches are found or the replacements would not change the file contents (the file already matches the replacement text) and the input and output filenames are the same, the existing file is not modified.

When the step completes, the following temporary macros are created or updated:

- **REPLACEINFILE\_MATCH\_COUNT** = The number of matches found in the file
- **REPLACEINFILE\_REPLACE\_COUNT** = The number of replacements made in the file

#### Replace tab

#### Text tab

*Note:* See the Files.bld sample for a project utilizing this action.

#### 3.4.13.1 Replace Tab

This tab of the Replace in File action specifies the file to be processed and options about matching to be performed.

**Input file:** The file to find text in. The entire file will be loaded into memory when the step is built.

**Output file:** The file to write the updated file contents with any replacements to (optional). The output file can be a different file. The existing file will be updated if this field is blank or the same as the input file. The folder specified in the output filename must already exist.

**Ensure output file is writeable:** If checked and the output file already exists with read-only attributes, the read-only attribute is removed before overwriting.

**Case-sensitive matching:** If unchecked, matches are case-insensitive; if checked the case must match when searching.

**Replace first match only:** If checked, only the first match is replaced; if unchecked, all matches are replaced.

**Treat characters as literals:** If checked, all text for searching and replacing is treated as literal text (no regular expressions).

*Options that apply only to non-literal (regex) replacements:*

**Dot does not match newline:** Causes . (period character) to not match newline characters (negates Perl /s modifier).

**Allow comments and ignore white space (Perl extended mode):** If checked, white space outside of character classes is (mostly) ignored and anything following an unescaped # character is treated as a comment.

**Binary Mode:** Treats the file as binary instead of text, performing ASCII text replacements on matching text within the binary file without converting to and from the file's encoding.

*Note:* Use caution when performing replacements in binary files, as this could render the file unreadable by the source application.

**Log file contents before and after replacements:** If checked, the file contents are logged before replacements are made and after (if any replacements made).

**Do not replace:** If checked, no replacements will be made, but any enabled logging will be displayed and a count of replacements that would have been made will be logged. Useful for debugging purposes.

**Encoding (code page):** The encoding or Windows code page that will be used when reading and writing the file (optional). If left blank, the system code page (CP\_ACP) will be used or the encoding specified by the BOM if the file contains a BOM (byte order mark). Other common encodings for files are UTF-8, Unicode (UTF-16 LE), Windows-1252, Cyrillic, Chinese, and other encodings or code pages supported by Windows (see the [Microsoft Global Development](#) web site and [MSDN Unicode topics](#) for more details).

### 3.4.13.2 Text Tab

This tab of the Replace in File action specifies the text to be found and replaced in the file.

**Text to find:** The text or regular expression to find. This can contain literal text or a regular expression specifying one or more items to match (see below).

*Notes:*

- To insert literal bracket [ ] and percent sign % characters within this field, they must be doubled up (just as in any other field), since these are normally interpreted by Visual Build as referencing script code and macros within a field.
- For regular expressions (non-literal) matching, . (dot) matches newline characters by default. Uncheck the Dot does not match newline option on the Replace tab to prevent . from matching newline characters and perform line-by-line matching.

**Replace matches with:** The text to replace any matches with.

For *literal text*, this can be a single search and replace value, with the *no regular expressions* field checked. To search and replace multiple literal values, enter each value on a separate line (the number of search and replace lines must match). For instance, to change all occurrences of 'fox' with 'cat' and 'dog' with 'beagle', the find expression

```
fox  
dog
```

and replace expression

```
cat
```

beagle

could be used. If a single-line string is provided in the find field and the replace text contains multiple lines, the value will be replaced with the entire multi-line value from the replace field.

For *regular expressions* (the literal option is unchecked on the Replace tab), each item within parentheses is grouped as a numbered expression, which can be matched in the replace field. Nested parentheses can be used to identify values within an overall matching value; each new parenthesis is numbered sequentially (starting at 1) in the order that the opening parenthesis characters are found in the string. Replacement groups are identified by ?n, and \$n will be replaced with the matching expression in the search expression.

More complex expressions, such as the one in the dialog above can also be defined (see the samples for more Replace in File samples). For example, these expressions change any DIALOG resources in a resource file to DIALOGEX and FONT "MS Sans Serif" to FONT "MS Shell Dlg":

Find text:

```
(^IDD([[^\s]]*\s+)\DIALOG\s+)|(^FONT([[^"]]*)\s*"MS Sans Serif"\s+)
```

Replace text:

```
(?1IDD$2DIALOGEX )(?3FONT$4"MS Shell Dlg")
```

**Text to append to file if no matches found:** If provided and no matches are found for the Text to find field, the text in this field will be appended to the output file. This can be useful for adding a missing value only if it is not already in the file (the find/replace fields are used to modify an existing value if found). This field is treated as literal text (after any macros or script expressions have been evaluated) and should not contain regular expression replacement variables.

### 3.4.14 Set File Attributes

The Set File Attributes action creates a step to set, clear, or list file attributes or timestamps. When built, the action finds all matching files (recursively searching for matches if specified) and updates the file attributes as specified (or logs the current attributes if not changed). If the attributes for a file are not changed, the current attributes are listed in brackets following the filename; if the attributes are changed, each attribute that was modified is listed, followed by a + or - to indicate whether the attribute was set or cleared.

When the step completes, the following temporary macros are created or updated:

- **SETFILEATTR\_COUNT** = The number of files whose attributes were modified
- **SETFILEATTR\_PROC\_COUNT** = The number of matching files processed by the step

#### Files Tab

#### Set Attributes Tab

#### Attributes Tab

##### 3.4.14.1 Files Tab

This tab of the Set File Attributes action specifies what files should be processed.

**Folder:** The source directory to find files in (required).

*Note:* You can also specify a path and filename or path and file mask in this field (instead of using the Include field).

**Files to process:** Used to specify files or folders to specifically include or exclude (optional).

**Process subdirectories:** Determines if subdirectories will be searched recursively matching files.

**Do not set attributes:** If checked, no file attributes will be modified, but any enabled logging will be displayed and a count of files that would have been modified will be logged. Useful for debugging purposes.

**Log files whose attributes are modified:** If checked, any files for which attributes are modified will be logged.

**Log files that are processed but not modified:** If checked, files that matched the include/exclude list but already matched the specified attribute settings will also be logged. Useful for listing existing attributes.

#### 3.4.14.2 Attributes Tab

This tab of the Set File Attributes action specifies the attributes to change.

**Read-only attribute:** Specifies a new value for the read-only attribute.

**Hidden attribute:** Specifies a new value for the hidden attribute.

**System attribute:** Specifies a new value for the system attribute.

**Archive attribute:** Specifies a new value for the archive attribute.

**Modify date+time:** Specifies a new value for the modified date/time (optional, not changed if blank).

**Create date+time:** Specifies a new value for the creation date/time (optional, not changed if blank).

**Access date+time:** Specifies a new value for the access date/time (optional, not changed if blank).

*Notes:*

- The date should be entered in the short date/time format as configured in the Windows Control Panel Regional settings.
- Use the system macro %DATETIME% to set the timestamp to the current date and time (touch).

#### 3.4.15 Sign Code

The Sign Code action creates a step to add a digital certificate to a file or to verify a file's digital certificate.

##### Sign Tab

##### Options Tab

##### Advanced Tab

##### Remote Tab

#### Steps to Automate Code Signing

When the step completes, the following temporary macros are created or updated:

- **SIGNCODE\_RESULT** = The command performed (0 = Sign, 1 = Timestamp, 2 = Verify) or -1 if



signing, the *Don't sign if already signed* option on the Options tab is checked, and the file was already signed

*Note:* Use with the Process Files action to sign multiple files.

### 3.4.15.1 Sign Tab

This tab of the Sign Code action specifies the file to be signed and other output options.

**File to sign:** The filename to sign (required). Must be a valid Windows executable (.exe, .dll, .ocx, .scr, .sys, .drv, .appx) or a .msi, .cab, .class, .js, or .vbs file.

**Command:** The command to perform on the file:

- *Sign* adds or replaces a code signing digital certificate and optionally timestamps the certificate.
- *Timestamp* updates the timestamp of the digital signature.
- *Verify* verifies that the file is signed with a valid digital signature.

**Append signature:** Appends the signature. If no primary signature is present, this signature is made the primary signature instead. This can be useful when dual-signing an executable with SHA1 and SHA256 certificates (sign first with the SHA1 certificate, then append the SHA2 certificate for highest compatibility with all Windows versions).

**Location:** The location to retrieve the public and private key information for the digital certificate (a PFX file or a certificate store).

**Store name/PFX File:** The name of the certificate store (defaults to "My" if not provided") or the PFX file for the PFX file location option.

**PFX password:** The password used to encrypt the PFX file (required and applies only for PFX File location option).

**Common name:** The common name of the certificate in the certificate store (optional, does not apply for PFX File option).

**Thumbprint:** The SHA1 hash thumbprint of the certificate to sign with (optional).

**Require only one matching valid certificate:** If unchecked, the action will find all valid certificates that satisfy all specified conditions and select the one that is valid for the longest time. If checked, the action expects to find only one matching, valid signing certificate.

For the Verify command, if this option is unchecked, all methods will be used to verify the file. First, the catalog databases are searched to determine whether the file is signed in a catalog. If the file is not signed in any catalog, the action attempts to verify the file's embedded signature. Unchecking this option is recommended when verifying files that may or may not be signed in a catalog. Examples of these files include Windows files or drivers.

**Timestamp server:** The URL of the timestamp server to use to timestamp the digital certificate (optional). The certificate will not be timestamped if this field is blank. Required for the timestamp command.

**RFC 3161 timestamp server:** If checked, the Timestamp server field specifies the URL of an RFC 3161-compatible timestamp server. If unchecked, the Timestamp server field specifies the URL of a legacy Authenticode timestamp server. Requires signtool.exe v6.2 or later.

**Information URL:** A URL to store in the certificate which provides more information about the file's content (optional).

### 3.4.15.2 Options Tab

This tab of the Sign Code action specifies additional signing options.

**Description:** Specifies a name that represents the contents of the signed file (optional).

**Signature hash algorithm:** The hash/digest algorithm to use when signing (optional). Values other than SHA1 requires signtool.exe v6.2 or later.

**Timestamp server hash algorithm:** The hash/digest algorithm for the timestamp server to use (optional, applies only for sign and timestamp commands). If a value other than SHA1 is specified, you must use an RFC 3161 timestamp server (check the related option on the Sign tab) that supports SHA2 certificates. Values other than SHA1 require signtool.exe v6.2 or later.

**Don't sign if already signed:** If checked and the file is already signed, it will leave the existing signature. If unchecked, the file will always be signed again.

**Retain original modification timestamp:** If checked, after signing the file, its modification date+time will be restored to the value it had before signing.

**Use Authenticode verification policy:** Applies only to the Verify command. If unchecked, uses the Windows Driver Verification Policy rather than the Authenticode Verification Policy.

**Verify all signatures:** Applies only to the Verify command. If checked, verifies all signatures in a file that includes multiple signatures.

**Verbose output:** If checked, the information about the file's digital certificate is logged.

**Minimal output:** If checked, logs no output on successful execution and minimal output for failed execution (mutually exclusive with Verbose option).

**Additional command-line options:** This action invokes the `signtool.exe` command-line program to sign, timestamp, or verify a file, and this option can be used to specify additional flags to pass to the tool.

**Override signtool executable filename:** If this field is blank, the action will attempt to automatically locate `signtool.exe` in a Microsoft Windows SDK installation or the Visual Build Tools path. If not found, either install the Tools option of the Microsoft Windows SDK or reinstall Visual Build and select *Full installation*.

### 3.4.15.3 Steps to Automate Code Signing

#### Steps to Automate Code Signing

##### 1. Obtain the Code Signing Tools

First, you need to obtain the necessary code signing tools. Several security-related tools will be required. If you have Visual Studio 2005 or later installed, these are already available (and added to the PATH of a Visual Studio Command Prompt).

Otherwise, download and install the Microsoft Windows SDK to install the necessary tools (only the Core SDK Tools need to be installed [about 40MB]). The required programs are installed to the Bin path under the Platform SDK install path.

Your digital certificate(s) can be viewed in the Windows certificate store using the Certificate Manager (`certmgr.exe`), which can also be launched from Internet Explorer (*Tools | Internet Options | Content | Certificates* for IE 6) or Outlook Express (*Tools | Options | Security | Digital IDs* for OE 6). The

Certificate Manager is also used to import and export certificates.

## 2. Obtain a Code Signing Certificate

The next step is to get a code signing certificate (or digital ID) from a certification authority (CA) such as [Ascertia](#), Comodo, GlobalSign, [Symantec](#), or [Thawte](#). You will need a Class 3 digital certificate for code signing.

During the sign-up process, a private key will be generated; make sure to mark the key as exportable. During this process, you will need to provide an email address, password, and challenge phrase, as well as additional information about your company. Save the private key to a local `.pvk` file and store it and the information associated with it in a secure location. Once your company information has been verified, the CA will issue the public portion of your digital certificate; you should save this to a local `.cer` file during the installation process and also store in a safe place.

*Note:* You can also create test certificates using [makecert.exe](#).

## 3. Convert the Certificate File Format

After receiving your code signing certificate (`.cer`), you need to convert it into a software publishing certificate (`.spc`) [cert2spc.exe](#). From a DOS/Command Prompt change to the path containing the `.cer` file and run:

```
cert2spc xyz.cer xyz.spc
```

Then you should convert your certificate (the public [`.spc`] and private [`.pvk`] portions) into a single Personal Information Exchange (`.pfx` or [PKCS #12](#)) file. This simplifies later steps and avoids incompatibility problems when using the certificate on different versions of Windows:

```
pvk2pfx -pvk xyz.pvk -pi <pvkpassword> -spc xyz.spc -pfx <xyz.pfx> -po  
<pfxpassword> -f
```

Store the `.pfx` file and the associated password in a secure location.

Finally, import your certificate into a certificate store on the build box by running `certmgr.exe`, clicking Import, entering the `.pfx` filename, the `.pfx` file password, and leaving *Enable strong private key protection* unchecked.

## 4. Add Code Signing to the Automated Build

Add a Sign Code step to your Visual Build project for each file to be signed (to sign multiple files, this can be simplified by using the Process Files action) by specifying information about the certificate in the certificate store or by providing the `.pfx` file and its associated password.

### 3.4.16 Transform XML Log

The Transform XML Log action creates a step to:

- Close the log tags in an XML log file (for attaching to an email during a build)
- Convert an XML log file into another format, such as an *HTML build report* or *RSS feed*, with optional filtering of builds and step output
- Apply an XSLT stylesheet to any XML document
- Format (indent) an XML document

*Note:* The Logging.bld sample demonstrates using this action (enable XML logging in application options).

When this action is built, the following logic is executed:

- 1) If the *Close tags* field is checked and if there are any missing step, build, or logs tags, the log file is copied and the closing tags added to the copy (the copy is deleted after the action finishes unless no stylesheet was specified).
- 2) If an XSLT stylesheet was specified,
  - 2a) If any custom filtering was specified on the Filter tab, the values are passed to the stylesheet (via its param tags).
  - 2b) The stylesheet is applied to the input document and written to the output file. If the default stylesheet is used for an XML log file, an HTML document is generated containing the specified build output and including summary and detail sections.
- 3) If *Format (indent)* is checked, the input file is formatted (indented).
- 4) If *Display output file* is checked, the output file is launched in the associated viewer (based on the file's extension).

## Transform tab

### Filter tab

### Parameters tab

#### Notes:

- This action requires that the [MSXML parser](#) (version 3.0 or later) be installed in order to generate its output (unless no XSLT stylesheet is specified).
- It is recommended that any active scanning anti-virus software be disabled on the build box, as this can interfere with Visual Build writing to its log file (and also slows down builds).

#### 3.4.16.1 Transform Tab

This tab of the Transform XML Log action specifies information about the file to be processed.

**Input file:** The source [XML](#) log file or document to process (required). This can be a Visual Build XML log file or any other valid XML document.

**Close input log file XML tags:** If checked, any missing step, build, or log elements are added to the input file before processing to ensure that it is a valid XML file. The changes are made on a copy of the input file so that the original log file is not affected, and the XSLT stylesheet (if specified) is then applied to the copy.

**Preserve white space:** If checked, preserves white space in the document.

**Do not validate DTD or schema:** If checked, the document will not be validated against a document type definition (DTD), schema, or schema cache.

**Do not resolve external definitions:** If checked, external definitions, resolvable namespaces, DTD external subsets, and external entity references will not be resolved at parse time, independent of validation.

**Prohibit inclusion of a DTD:** Determines whether inclusion of a DTD is allowed or prohibited.

**Enable XSLT script code execution:** Determines whether `<msxsl:script>` element functionality is allowed.

**Enable document function:** Determines whether the document function is allowed.

**Use inline schemas for validation:** Determines whether inline schemas should be processed for validation.

**XSLT stylesheet:** The [XSLT](#) stylesheet to apply to the document (optional). Leave blank if no stylesheet should be applied (to close the XML log tags on the log file). Some default XSLT stylesheets are provided in the Style subfolder of the installation path:

- `TransformLog.xslt` to convert the log to an HTML report with a summary and detail section for the specified builds.
- `RSS.xslt` to convert the log to an RSS feed.

Also, any valid stylesheet can be specified (although in order for filtering of log output to function correctly, the stylesheet's `param` tags must be supplied and used in the same manner as the default stylesheets).

*Note:* The default stylesheets can be modified by the user (to change the document layout, fonts, colors, etc.), but any changes should be saved under a different filename, since the default stylesheets will be overwritten when updates to Visual Build are installed or if it is re-installed.

**Include nested (chained) build steps:** Determines whether step output for nested builds will be included in the HTML report.

**Output file:** The destination filename to generate (required).

*Note:* To make the XML or HTML build log file accessible to other users, simply map an IIS or other web server virtual directory to the path containing the log or HTML files, and the log files can be accessed via any web browser.

**Format (indent) document:** Whether to format (indent) the XML document.

**Encoding:** The encoding for the formatted XML file (applies only when formatting).

**Display output file:** Indicates that the resulting output file should be launched in its associated application.

### 3.4.16.2 Filter Tab

This tab of the Transform XML Log action specifies filtering which builds and/or steps to include in the output document. Builds and steps can be included or excluded by status (for instance, only failed builds), and the number of recent builds to include can also be specified.

*Note:* The fields on the Filter tab only apply if an XSLT stylesheet is specified on the Transform tab. If the action is being used to transform another XML document (not a Visual Build log file), leave these fields at their default values so that no parameters will be applied to the stylesheet.

**Builds to include:** This field can be used to include all or to limit the number of builds to include in the output file. A log file can contain an unlimited number of builds, but this field can be used to only include a maximum number of most recent builds.

**Filter builds by completion status:** Use to only include builds of one or more specific completion status codes.

**Filter steps by build status:** Use to only include steps of one or more specific build status codes. Uncheck all statuses to exclude the detail section of the log report.

**Filter step output:** Use to filter the level of log messages that are included in the output and the length of output to include (if maximum output length is specified, each step's output is truncated at the maximum length specified. Use a length of 0 to suppress the display of all step output).

### 3.4.16.3 Parameters Tab

This tab of the Transform XML Log action specifies custom XSL parameters.

**Parameter name:** Names of parameters to pass. Each parameter name and value is passed to the XSL stylesheet.

**Parameter value:** Values of parameters to pass.

Enter a parameter name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Mode:** Specifies the XSLT processing mode to use (optional).

### 3.4.17 Write File

This action creates a step to create a text file or append text to a file.

**Filename:** The filename to write the text to. Any non-existent subdirectories will be created before creating the file.

**Text:** The text to write to the file.

**Append to file:** If checked, the file will be appended to if it already exists. If unchecked, the file will be replaced if it exists.

**Ensure file is not read-only:** Removes the read-only attribute of the file before overwriting or appending.

**Encoding (code page):** The encoding or Windows code page that will be used when writing the file (optional). If left blank, the system code page (CP\_ACP) will be used. Other common encodings for files are UTF-8, Unicode (UTF-16 LE), Windows-1252, Cyrillic, Chinese, and other encodings or code pages supported by Windows (see the [Microsoft Global Development](#) web site and [MSDN Unicode topics](#) for more details).

**Write BOM:** If checked, writes a byte order mark to the beginning of the file. Applies only when creating or replacing a file and using an encoding of UTF-8 or Unicode.

*Note:* Bracket characters [ and ] and the percent sign character % within step fields have special meaning in a step field.

### 3.4.18 Write INI

This action creates a step to create, update, or delete values in an INI file. The Read INI action can be used to read values from an INI file.

**Filename:** Name of the INI file to update (required). Any non-existent subdirectories will be created before creating the file. If no path is specified, the file will be created/updated in the Windows folder ( %WINDIR% system macro).

**Section:** Section of the file to update (required).

**Value name:** Value name to create or delete (optional). If the value name is empty, the entire section will be deleted if delete is checked.

**Delete the value:** Deletes the value if checked, otherwise creates/updates the value.

**Value data:** The string to set the given value to (optional).

**Ensure file is not read-only:** Removes the read-only attribute of the file before overwriting or appending.

**Fail step if file, section, or value not found:** If checked and the file, section, or value doesn't exist, the step will fail. If unchecked, the file, section, and/or value will be created if not found.

*Notes:*

- Bracket characters [ and ] and the percent sign character % within step fields have special meaning in a step field.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.4.19 Write XML

This action creates a step to create or update element or attribute values in an XML file, saving the file only if changes were made.

#### File Tab

#### Namespaces Tab

#### XPath Tab

*Notes:*

- This action requires MSXML version 3, 4 or 6 to be installed (the highest available version will be used).
- See the XML.bld sample for sample usage.
- The Read XML action can be used to read values from an XML file.

#### 3.4.19.1 File Tab

This tab of the Write XML action specifies information about the file to be processed.

**Filename:** Name of an existing XML file to update (required).

**Do not validate against DTD or schema:** If checked, the document will not be validated against a document type definition (DTD), schema, or schema cache.

**Do not resolve external definitions:** If checked, external definitions, resolvable namespaces, DTD external subsets, and external entity references will not be resolved at parse time, independent of validation.

**Prohibit inclusion of a DTD:** Determines whether inclusion of a DTD is allowed or prohibited.

**Preserve white space:** If checked, preserves white space in the document.

**Enable XSLT script code execution:** Determines whether <msxsl:script> element functionality is allowed.

**Enable document function:** Determines whether the document function is allowed.

**Use inline schemas for validation:** Determines whether inline schemas should be processed for validation.

**Make file writeable:** If checked and the file is read-only on disk, the read-only attribute will be

removed before saving.

### 3.4.19.2 Namespaces Tab

This tab of the Write XML and Read XML action specifies namespace aliases and URIs for the query (optional). Enter a namespace alias and namespace URI in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list. Each set of values is added as a selection namespace declaration for the query (i.e., `xmlns:alias='namespaceURI'`).

### 3.4.19.3 XPath Tab

This tab of the Write XML action specifies the XPath expression and text to update or create.

**XPath expression:** An XPath expression that identifies the node(s) to update (required).

**Force XPath query language for MSXML v3:** If checked and MSXML v3 is used, the query language will be set to XPath (XPath is always used by newer versions of MSXML).

**Attribute to update:** The name of an attribute within the specified node(s) to create or update (optional). If blank, the text for the node(s) matching the XPath expression will be updated.

**Create element/attribute:** If the specified element or attribute does not exist and this option is checked, it will be created. If it does not exist and this option is unchecked, an error will occur.

**Delete element/attribute:** Deletes the specified element or attribute if found.

**Text to assign:** The text to set the matching element or attribute value to (optional).

**Append:** Whether to append to or replace any existing value.

**Update all matching nodes:** If checked, all matching nodes will be updated; if unchecked, only the first matching node will be updated.

*Note:* Bracket characters [ and ] and the percent sign character % within step fields have special meaning in a step field.

## 3.5 Help Authoring

### 3.5.1 Doc-O-Matic

This action creates a step to compile a toolfactory Doc-O-Matic project file. From the inputs, the Doc-O-Matic command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

#### Project Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with versions 5, 6, and 7 of Doc-O-Matic and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all



hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.1.1 Project Tab

This tab of the Doc-O-Matic action configures information about the help project being compiled.

**Project filename:** Specifies the Doc-O-Matic project file to compile (required).

**Symbol ID list file to generate:** Generates a symbol ID list from the project instead of compiling the project (optional).

**Configuration to build:** Indicates which configuration to generate (optional; builds all configurations if blank).

**Do not build:** Check for errors only if checked.

**Do not show documentation after build completes:** Generated documentation is not shown after build completes if checked.

**Treat warnings as errors:** If checked and the build output contains warnings, the step will fail.

**Log level:** Specifies the level of log output to generate.

### 3.5.1.2 Options Tab

This tab of the Doc-O-Matic action configures additional options.

**Load specific filter DLL:** Specifies a filter DLL to load (optional).

**Enter any additional options:** This action invokes the Doc-O-Matic application to compile the help file outputs, and this option can be used to specify additional flags to pass it. See *Command line options* in the Doc-O-Matic help index for the available options.

**Override the Doc-O-Matic executable filename:** If this field is empty, the action automatically locates the Doc-O-Matic command-line executable. This can be overridden by specifying the executable filename here.

## 3.5.2 Doc-To-Help

This action creates a step to compile a Doc-To-Help project file. From the inputs, the Doc-To-Help command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

### Project Tab

### Options Tab

### Advanced Tab

### Remote Tab

This action has been tested with Doc-To-Help 2006 thru 2014 and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.2.1 Project Tab

This tab of the Doc-To-Help action configures information about the help project being compiled.

**Project filename:** Specifies the Doc-To-Help project file to compile (required).

**Build mode:** Determines the build behavior (required; make => updates help targets; build => rebuilds help targets; compileall => compiles all project files).

**Output formats:** Indicates one or more output formats to generate (required).

**Log progress messages:** If checked, progress messages will be logged during compilation.

### 3.5.2.2 Options Tab

This tab of the Doc-To-Help action configures additional options.

**Additional command-line options:** This option can be used to specify additional flags to pass to the Doc-To-Help compiler. Search for *Command line* in the Doc-To-Help help for the available options.

**Override compiler executable filename:** If this field is empty, the action attempts to automatically locate the Doc-To-Help command-line executable. This can be overridden by specifying the executable filename here.

## 3.5.3 Document! X

This action creates a step to compile an Innovasys Document! X project file. From the inputs, the Document! X command-line executable is constructed and called when the action is built.

### Project Tab

### Options Tab

### Advanced Tab

### Remote Tab

This action has been tested with versions 5 thru 2015 of Document! X and may also work with other versions.

#### Notes:

- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- For v2008 and earlier, this action requires that `DX5CommandLine.exe` (provided by Innovasys with an Automation license) be installed in the same path as the Document! X IDE (`DXIDE5.exe`), or that the path+filename to this executable be provided in the Override field on the Options tab.

### 3.5.3.1 Project Tab

This tab of the Document! X action configures information about the project being compiled.

**Project filename:** Specifies the Document! X project file to compile (required).

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all

hidden actions, right-click in the Actions pane and choose *Show Hidden*.

*Note:* The following options apply only for v2010 and later:

**Build configuration:** The build configuration/profile to compile (optional, first profile will be built if not specified).

**Compile .NET v4:** Runs under .NET Framework version 4 (use to document any projects containing .NET 4 assemblies).

**Ignore warnings:** Do not fail the step if only warnings are reported.

*Note:* The following options apply only for v2008 and earlier:

**Run silently:** Prevent display of message boxes and dialogs.

**Show progress dialog:** Shows a progress dialog while compiling.

**Show results dialog:** Display a results dialog after compiling.

**Output directory:** Overrides the default output directory (optional).

**Output filename:** Overrides the default output filename (optional).

### 3.5.3.2 Options Tab

This tab of the Document! X action configures additional options.

**Additional command-line options:** This option can be used to specify additional flags to pass to the Document! X compiler.

**Override compiler executable filename:** If this field is empty, the action automatically locates the Document! X command-line executable by finding the executable associated with the Document! X project filename extension (.dxp). This can be overridden by specifying the executable filename here.

*Note:* In order to locate automatically, this action requires that `DX5CommandLine.exe` (for Document! X v5, this is provided separately by Innovasys with an Automation license) be installed in the same path as the Document! X IDE (`DXIDE5.exe`), or that the path+filename to this executable be provided in the Override field.

### 3.5.4 Dr.Explain

This action creates a step to compile a Dr.Explain project file. From the inputs, the Dr.Explain command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with versions 3 thru 5 of Dr.Explain and may also work with other versions.

**Filename:** Specifies the Dr.Explain project file to compile (required).

**Formats:** Specifies one or more formats to compile (required). Select or enter a format in the drop-down list and click Insert to add; select a format in the grid and click Delete to remove.

**Additional command-line options:** This option can be used to specify additional flags to pass to the Dr.Explain compiler (optional).

**Override compiler executable filename:** If this field is empty, the action automatically locates the

Dr.Explain command-line executable by finding the executable associated with the Dr.Explain project filename extension. This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.5 Fast-Help

This action creates a step to compile a Fast-Help project file. From the inputs, the Fast-Help command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with versions 4 through 8 of Fast-Help and may also work with other versions.

**Project filename:** Specifies the Fast-Help project file to compile (required).

**Output formats:** Indicates one or more output formats to generate (optional).

**Additional command-line options:** This option can be used to specify additional flags to pass to the Fast-Help compiler. See *Command line* in the Fast-Help help index for the available options.

**Override compiler executable filename:** If this field is empty, the action attempts to automatically locate the Fast-Help command-line executable. This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.6 Flare

This action creates a step to compile a MadCap Flare project file. From the inputs, the Flare command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with versions 1 through 11 of Flare and may also work with other versions.

**Filename:** Specifies the Flare project file to compile (required).

**Target:** Indicates a specific target to generate (optional, all targets are built if blank).

**Build target:** Indicates a batch target to generate (requires v6 or later). Mutually exclusive with Target field.

**Additional command-line options:** This option can be used to specify additional flags to pass to the Flare compiler. See *Command line* in the Flare help index for the available options.

**Override compiler executable filename:** If this field is empty, the action automatically locates the Flare command-line executable by finding the executable associated with the Flare project filename extension (`.flprj`). This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.7 Help & Manual

This action creates a step to compile a [Help & Manual](#) project file. From the inputs, the Help & Manual command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

#### Project Tab

#### Include Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with versions 3 thru 7 of Help & Manual and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.5.7.1 Project Tab

This tab of the Help & Manual action configures information about the help project being compiled.

**Filename:** Specifies the Help & Manual project file to compile (required).

**Output formats:** Indicates which output format(s) to generate and (optionally) the filename for each format. At least one output format should be checked. Relative paths are relative to the path of the help project file.

**PDF Manual template file:** Specifies the name of the PDF manual template. This parameter is optional and only enabled when the Adobe PDF output format is checked. If not provided, the template that is specified for PDF output in project properties is used. Relative paths are relative to the path of the help project file.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.5.7.2 Include Tab

This tab of the Help & Manual action configures additional include options.

**Include options:** Specifies the include options, similar to the include options in the Help & Manual Make dialog box. If you do not specify this value, the default include options are used. The All and MS Help 2.0 options are specific to Help & Manual version 4+.

**Only include topics and chapters with status of complete:** All topics with any other status are excluded from published output.

**Table of contents:** Specifies the TOC to use for publishing projects with multiple TOCs (optional, applies only to Help & Manual version 7+).

**Additional user-defined options:** Use to specify additional user-defined include options (optional,

applies only to Help & Manual version 4+).

**Text variables file:** Overloads text variables defined in the help project with values from an external file (optional). The text file must contain variable values in the form of VARIABLENAME=VALUE. A Write File action could be used to generate this file. Relative paths are relative to the path of the help project file.

### 3.5.7.3 Options Tab

This tab of the Help & Manual action configures additional Help & Manual options.

**Publishing tasks to execute:** Specifies one or more tasks/actions to execute, one per line (requires v6+).

**Override the Help & Manual executable filename:** If this field is empty, the action automatically locates the H&M command-line executable by finding the executable associated with the Help & Manual project filename extension (.hm3 for v3 and .hmx, for v4, and .hmxz or .hmxp for v5 and later). This can be overridden by specifying the executable filename here.

**Enter any additional options:** This action invokes the Help & Manual application to compile the help file outputs, and this option can be used to specify additional flags to pass it. See *Command line options* in the Help & Manual help index for the available options.

**Log console output:** If this option is checked (the default), any output written to the console by the Help & Manual executable will be captured and logged to the Output pane (and log file if file logging is enabled). If this option is unchecked, the Help & Manual flag to generate a log file will be specified, and the log file's contents will be logged. *Note:* Help & Manual v5.3 and later log compiler output to the console; uncheck this option for previous versions to capture the log file output.

**Parse output and fail step if warnings found:** If checked, after the step completes, the Help & Manual log file will be loaded and examined, and if any [Warning] lines are found, the step will be marked as failed.

**Show debug info:** Display a message window before outputting showing the batch commands and whether they have been recognized properly. Use for troubleshooting if your command line doesn't work as expected (requires Help & Manual v5 or later).

**Don't delete temporary files:** Don't delete the temporary source directories and files generated when compiling HTML Help, Winhelp, and Visual Studio Help / MS Help 2.

## 3.5.8 Helpinator

This action creates a step to compile a Helpinator project file. From the inputs, the Helpinator command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

### Project Tab

### Options Tab

### Advanced Tab

### Remote Tab

This action has been tested with versions 1 thru 3 of Helpinator and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be

displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.8.1 Project Tab

This tab of the Helpinator action configures information about the help project being compiled.

**Filename:** Specifies the Helpinator project file to compile (required).

**Output formats:** Indicates which output format(s) to generate (required).

**Output folder:** Specifies the full output folder (required).

### 3.5.8.2 Options Tab

This tab of the Helpinator action configures additional Helpinator options.

**Compiler variables:** Specifies project variable-value pairs (optional). Enter a definition variable name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Additional options:** This action invokes the Helpinator application to compile the help file outputs, and this option can be used to specify additional flags to pass it.

**Override the Helpinator executable filename:** If this field is empty, the action automatically locates the Helpinator command-line executable by finding the executable associated with the Helpinator project filename extension. This can be overridden by specifying the executable filename here.

## 3.5.9 HelpMaker

This action creates a step to compile a HelpMaker project file. From the inputs, the HelpMaker command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with version 7 of HelpMaker and may also work with other versions.

**Filename:** Specifies the HelpMaker project file to compile (required).

**Formats:** Specifies one or more formats to compile (required). Select or enter a format in the drop-down list and click Insert to add; select a format in the grid and click Delete to remove.

**Additional command-line options:** This option can be used to specify additional flags to pass to the HelpMaker compiler (optional).

**Override compiler executable filename:** If this field is empty, the action automatically locates the HelpMaker command-line executable by finding the executable associated with the HelpMaker project filename extension. This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

## 3.5.10 HelpNDoc

This action creates a step to compile a HelpNDoc project file. From the inputs, the HelpNDoc command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

### Project Tab

## Output Tab

## Output (More) Tab

## Options Tab

## Advanced Tab

## Remote Tab

This action has been tested with versions 3 and 4 of HelpNDoc and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.10.1 Project Tab

This tab of the HelpNDoc action configures information about the help project being compiled.

**Filename:** Specifies the HelpNDoc project file to compile (required).

**Run silently:** Don't prompt for user input if checked.

**Builds to generate:** Specifies the builds to generate (optional, one per line). This option applies only to HelpNDoc v3.6 and later.

### 3.5.10.2 Output Tab

*Note:* These options apply only to version prior to HelpNDoc 3.6.

This tab of the HelpNDoc action configures additional output options. If not specified, the project default settings will be used.

**Set the CHM generation flag for the project:** Causes CHM output to be generated.

**CHM generator output file:** Sets the CHM output filename (optional).

**CHM generator template name:** Sets the CHM template name (optional).

**Unset the CHM generation flag for the project:** Causes CHM output to not be generated.

**Set the HTML generation flag for the project:** Causes HTML output to be generated.

**HTML generator output file:** Sets the HTML output filename (optional).

**HTML generator template name:** Sets the HTML template name (optional).

**Unset the HTML generation flag for the project:** Causes HTML output to not be generated.

### 3.5.10.3 Output (More) Tab

*Note:* These options apply only to version prior to HelpNDoc 3.6.

This tab of the HelpNDoc action configures additional output options.



**Set the PDF generation flag for the project:** Causes PDF output to be generated.

**PDF generator output file:** Sets the PDF output filename (optional).

**PDF generator template name:** Sets the PDF template name (optional).

**Unset the PDF generation flag for the project:** Causes PDF output to not be generated.

**Set the Word generation flag for the project:** Causes Word output to be generated.

**Word generator output file:** Sets the Word output filename (optional).

**Word generator template name:** Sets the Word template name (optional).

**Unset the Word generation flag for the project:** Causes Word output to not be generated.

#### 3.5.10.4 Options Tab

This tab of the HelpNDoc action configures additional HelpNDoc options.

**Save the settings to the project:** If checked, any settings configured on other tabs will be saved permanently to the project file.

**Additional options:** This action invokes the HelpNDoc application to compile the help file outputs, and this option can be used to specify additional flags to pass it. See *Usage from the command line* in the HelpNDoc help for the available options.

**Override the compiler executable filename:** If this field is empty, the action automatically locates the HelpNDoc command-line executable by finding the executable associated with the HelpNDoc project filename extension. This can be overridden by specifying the executable filename here.

**Variable names/values:** Specifies project variable-value pairs (optional). Enter a definition variable name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.5.11 HelpScribble

This action creates a step to compile a JGSoft HelpScribble project file. From the inputs, the HelpScribble command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

##### Project Tab

##### Options Tab

##### Advanced Tab

##### Remote Tab

This action has been tested with version 7 of HelpScribble and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.11.1 Project Tab

This tab of the HelpScribble action configures information about the help project being compiled.

**Project filename:** Specifies the HelpScribble project file to compile (required).

**Output formats:** Indicates the output format(s) to generate (optional).

**Main HTML file:** Specifies a complete path to the main HTML file of the exported web help (for Web format only).

### 3.5.11.2 Options Tab

This tab of the HelpScribble action configures additional options.

**Build tags:** If you have specified Build Tag properties for certain topics, you can select which help topics should be included in the compiled help file using this field (optional). If empty, all topics will be included. To include only certain topics, list the build tags you want, one per line. Only topics that have these build tags in their build tag property (and topics for which you left the build tag property blank) will be included in the compiled help file.

**Additional command-line options:** This option can be used to specify additional flags to pass to the Flare compiler. See *Command line* in the HelpScribble help index for the available options.

**Override compiler executable filename:** If this field is empty, the action automatically locates the HelpScribble command-line executable by finding the executable associated with the HelpScribble project filename extension (.hsc). This can be overridden by specifying the executable filename here.

## 3.5.12 HelpStudio

This action creates a step to compile an Innovasys HelpStudio project file. From the inputs, the HelpStudio command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

### Project Tab

### Options Tab

### Advanced Tab

### Remote Tab

This action has been tested with versions 2, 3, and 2011 thru 2015 of HelpStudio and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.12.1 Project Tab

This tab of the HelpStudio action configures information about the help project being compiled.

**Project filename:** Specifies the HelpStudio project file to compile (required).

**Build profile:** Indicates the build profile to build (optional, defaults to first build profile in the project).

**Build booklet:** Builds the specified booklet name (optional).

**Run silently:** Prevent display of message boxes.

**Show results dialog:** Display a Build Results dialog after compiling.

**Show progress dialog:** Shows a progress dialog while compiling.

**Output directory:** Overrides the default output directory (optional).

**Output filename:** Overrides the default output filename (optional).

### 3.5.12.2 Options Tab

This tab of the HelpStudio action configures additional options.

*Note:* The following two options apply only for v2011 and later:

**Build configuration:** The build configuration/profile to compile (optional, first profile will be built if not specified).

**Compile .NET v4:** Runs under .NET Framework version 4 (use to document any projects containing .NET 4 assemblies).

**Do not build Document! X context:** If checked, the Document! X context will not be generated.

**Additional command-line options:** This option can be used to specify additional flags to pass to the HelpStudio compiler.

**Override compiler executable filename:** If this field is empty, the action automatically locates the HelpStudio command-line executable by finding the executable associated with the HelpStudio project filename extension (.hsp). This can be overridden by specifying the executable filename here.

### 3.5.13 HyperText Studio

This action creates a step to compile a HyperText Studio project file. From the inputs, the HyperText Studio command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with version 5 of HyperText Studio and may also work with other versions.

**Filename:** Specifies the HyperText Studio project file to compile (required).

**Targets:** Specifies one or more targets to generate (optional, the default target is built if empty). Select or enter a target in the drop-down list and click Insert to add; select a target in the grid and click Delete to remove.

**Additional command-line options:** This option can be used to specify additional flags to pass to the HyperText Studio compiler (optional).

**Override compiler executable filename:** If this field is empty, the action automatically locates the HyperText Studio command-line executable by finding the executable associated with the HyperText Studio project filename extension (.hts). This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.5.14 RoboHelp

This action creates a step to compile a RoboHelp project file. From the inputs, the RoboHelp command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

#### Project Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with versions 6 thru 12 (2015) of RoboHelp and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.5.14.1 Project Tab

This tab of the RoboHelp action configures information about the help project being compiled.

**Project filename:** Specifies the RoboHelp project file to compile (required).

**Layouts:** Specifies one or more layouts to generate (optional, the primary layout is compiled if empty). Select or enter a layout in the drop-down list and click Insert to add; select a layout in the grid and click Delete to remove.

**Output folder:** Indicates the output format(s) to generate (optional).

**Display layout and publishing destination names:** Specifies a complete path to the main HTML file of the exported web help (for Web format only).

**Generate all layout sets for batch generation:** Specifies a complete path to the main HTML file of the exported web help (for Web format only).

#### 3.5.14.2 Options Tab

This tab of the RoboHelp action configures additional options.

**Additional command-line options:** This option can be used to specify additional flags to pass to the RoboHelp compiler.

**Override compiler executable filename:** If this field is empty, the action automatically locates the RoboHelp command-line executable by finding the executable associated with the RoboHelp project filename extension (.xprj). This can be overridden by specifying the executable filename here.

## 3.6 Installers

### 3.6.1 Advanced Installer

This action creates a step to compile an [Advanced Installer](#) setup project.

#### Project tab

#### Properties tab

#### Options tab

#### Advanced Tab

#### Remote Tab

This action has been tested with versions 3 thru 12 of Advanced Installer and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.1.1 Project Tab

This tab of the Advanced Installer action specifies the installer project to be compiled and other options.

**Project filename:** The filename of the project to compile (required).

**Package name:** Sets the name of the package in the project file (optional).

**Builds to perform:** List of builds that should be performed (optional).

#### 3.6.1.2 Properties Tab

This tab of the Advanced Installer action specifies installer project properties to be set.

**Product version:** Set the product version in the project file to the specified value or the version of the specified file (optional).

**Do not generate new product code:** When setting the product version, a new Product Code will be generated if the new version is different from the old one, unless this option is checked.

**Property names/values:** Defines property name-value pairs in the project file (optional). Enter a variable name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.6.1.3 Options Tab

This tab of the Advanced Installer action specifies additional options.

**Do not build project:** If checked, the project will not be compiled, and changes to the project specified on the Project tab will be processed.

**Force build:** If checked, builds the project even if up-to-date (does not apply with no build).

**Commands to execute:** Specifies one or more commands to perform, one per line (optional). The available commands are documented in the Advanced Installer help file under *Using Advanced Installer -> Command Line -> Editing*. Additionally, these additional commands can be specified:

- Save - Saves the project.
- Build - Builds the project.
- Rebuild - Performs a clean rebuild of the project.
- ResetSig - Resets the digital signature. The effect of this command is equivalent to unchecking the option "Sign the package" in the Digital Signature page. This is useful for instance in release candidate builds when signing the package and/or installation files is not required.

Example:

```
SetVersion 1.2
AddFile APPDIR\MyFolder C:\Folder\File.txt
SetProperty TESTPROP="TestValue"
save
rebuild
```

**Ignore errors when executing commands:** If checked, all errors are displayed when executing commands, but the execution of the commands file is not interrupted.

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to the Advanced Installer command-line compiler (optional).

**Compiler executable filename:** Use this field to specify the Advanced Installer compiler executable if the action is unable to locate it or multiple versions are installed (optional). If not specified, the executable is located automatically.

### 3.6.2 DeployMaster

This action creates a step to compile a DeployMaster project. The project filename can be entered or browsed to and the executable can be located automatically.

**Project filename:** The filename of the DeployMaster project to compile (required).

**Hide console window:** Hides the console window from displaying (applies to v6 only when building with the console app).

**Verbose output:** If checked, logs all output to the console. If unchecked, only headline messages, errors, and warnings are logged (applies to v6 only when building with the console app).

**Override compiler executable filename:** Determines if the compiler is located automatically or manually. If unchecked, the full and filename to the compiler executable must be entered in the following field. Use if multiple versions of the compiler are installed or to specify the executable file if the action is unable to.

This action has been tested with versions 2 thru 4 of DeployMaster and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.3 ExpertInstall / QuickInstall / Tarma Installer / Tarma InstallMate

This action creates a step to compile a Tarma InstallMate, Installer, ExpertInstall, or QuickInstall setup project.

## Project tab

## Options tab

## Advanced Tab

## Remote Tab

This action has been tested with versions 7 and 9 of Tarma InstallMate, version 5 of Tarma Installer, version 3 of Tarma ExpertInstall, and version 2 of Tarma QuickInstall and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.3.1 Project Tab

This tab of the ExpertInstall / QuickInstall / Tarma Installer / Tarma InstallMate action specifies the installer project to be compiled and other options.

**Project filename:** The filename of the project to compile (required).

**Configuration:** Specifies the configuration to build (optional, blank for default).

**Check only:** Checks the project without building if checked.

**Do not sign installer:** If checked, does not sign the installation package, even if the build configuration has its Sign after build attribute checked. This option is useful to avoid user interaction during a batch build.

**Log file:** Specifies the log file to write compilation output to (optional, reads from stdout if blank).

**Variable names/values:** Defines variable name-value pairs to set (optional). Enter a variable name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.6.3.2 Options Tab

This tab of the ExpertInstall / QuickInstall / Tarma Installer / Tarma InstallMate action specifies additional options.

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to the Tarma command-line compiler (optional).

**Compiler executable filename:** Use this field to specify the Tarma InstallMate, Installer, ExpertInstall, or QuickInstall compiler executable if the action is unable to locate it or multiple versions are installed (optional). If not specified, the executable is located automatically.

## 3.6.4 Inno Setup

This action creates a step to compile an [Inno Setup](#) script. The project filename can be entered or browsed to and the executable can be located automatically or manually. From these inputs, the action invokes the Inno Setup command-line compiler (iscc.exe) to create the installation executable.

## Project tab

## Options tab

### Advanced Tab

#### Remote Tab

This action has been tested with versions 2 thru 5 of Inno Setup and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.4.1 Project Tab

This tab of the Inno Setup action specifies the installer project to be compiled.

**Filename:** The filename of the Inno Setup script to compile (required).

**Quiet mode:** Quiet compile (print only error messages). Applies only to Inno Setup 5.05 and later.

**Output path:** Specify an output path for the executable (overriding any OutputDir setting in the script). Applies only to Inno Setup 5.05 and later.

**Output filename:** Specify an output filename for the executable (overriding any OutputBaseFilename setting in the script). Applies only to Inno Setup 5.06 and later.

**Preprocessor definition names/values:** Emulates an Inno Setup Preprocessor #define directive. Enter a definition name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

*Note:* This option applies only when the Inno Setup Preprocessor is installed.

#### 3.6.4.2 Options Tab

This tab of the Inno Setup action specifies additional options.

**Additional options:** Used to specify any additional options to the Inno Setup compiler or preprocessor (optional).

**Automatically locate the Inno Setup compiler executable:** Determines if the compiler is located automatically or manually. If unchecked, the full and filename to the compiler executable must be entered in the following field. Use if multiple versions of the compiler are installed or to specify the executable file if the action is unable to.

If auto-locate is checked, the Inno Setup compiler is located by:

- 1) Determine the executable associated with the Compile verb for the extension of the file in the Filename field.
- 2) Change the exe filename to `iscc.exe`.

Inno Setup scripts normally have a `.iss` extension, and that extension's Compile verb must be associated with Inno Setup for the above logic to work. In this case, manually specify the full path and filename for the compiler (this can be stored in a global macro to use across multiple steps or projects).

**Use ISPP-compatible command-line:** If checked (the default), the script filename will be placed before any other parameters for compatibility with the Inno Setup Preprocessor, which requires the



script filename first. Check this option if the Inno Setup Preprocessor (ISPP) is installed.

### 3.6.5 InstallAnywhere

This action creates a step to compile an [InstallAnywhere](#) setup script. The project filename can be entered or browsed to, the executable can be located automatically or manually, and the compile options configured from this action. From these inputs, the InstallAnywhere command-line call is generated and run when the step is built.

#### Project tab

#### Options tab

#### Advanced Tab

#### Remote Tab

This action has been tested with version 5.5 thru 2015 of InstallAnywhere (Standard and Enterprise) and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.5.1 Project Tab

This tab of the InstallAnywhere action specifies the install project to be compiled and which platforms to compile for.

**Filename:** The filename of the project to compile (required).

**Properties file:** Use the options from the specified build properties file (optional). If this field is provided, all other options configured in the action will be ignored (see the InstallAnywhere documentation on the available options).

**Platforms:** Check the platform(s) to build installers for. The appropriate flags will be added to the command-line for each platform. If no platforms are checked, the settings stored in the install script will be used.

#### 3.6.5.2 Options Tab

This tab of the InstallAnywhere action specifies additional options.

**Build Web Installers:** Indicates whether web installers will be generated.

**Build CD Installers:** Indicates whether CD installers will be generated.

**Optimize by platform:** Indicates whether optimizations by platform will be performed.

**Build merge modules:** Indicates whether merge modules will be generated.

**Additional options for to pass to InstallAnywhere:** Use this field to enter any additional command-line flags to be passed to the InstallAnywhere command-line compiler.

**Automatically locate the executable:** Determines if the compiler is located automatically or manually. If unchecked, the full and filename to the compiler executable must be entered in the following field. Use if multiple versions of the compiler are installed or to specify the executable file if

the action is unable to locate it. If auto-locate is checked, the compiler is located by combining the path of the executable associated with the extension of the file in the Filename field with `Build.exe`.

### 3.6.6 InstallAnywhere.NET

This action creates a step to compile an InstallAnywhere.NET installer project.

#### Project tab

#### Options tab

#### Advanced Tab

#### Remote Tab

This action has been tested with version 3.0 and 3.1 of InstallAnywhere.NET and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.6.1 Project Tab

This tab of the InstallAnywhere.NET Action action specifies the install project to be compiled and other compile options.

**Project filename:** The filename of the project to compile (required).

**Build configuration:** Specifies the build configuration to use (optional). If blank, the default configuration will be built.

**Suppress logo:** Prevents the program logo from being logged.

**Verbose log output:** Logs detailed information about the compilation.

**Log warnings**

**Log errors**

**Save project after build**

#### 3.6.6.2 Options Tab

This tab of the InstallAnywhere.NET Action action specifies additional options.

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to the InstallAnywhere.NET command-line compiler (optional).

**Compiler executable filename:** Use this field to specify the InstallAnywhere.NET compiler executable if the action is unable to locate it or multiple versions are installed (optional). If not specified, the executable is located automatically.

### 3.6.7 InstallAware

This action creates a step to compile an [InstallAware](#) setup project.

## Project tab

## Optimizations tab

## Options tab

## Advanced Tab

## Remote Tab

This action has been tested with versions 3 (Professional & Enterprise), 2005 (Developer, Studio, & Architect), 5 thru 18 (Developer, Studio, & Studio Admin), X2, 2012, NX, and X3 of InstallAware and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.7.1 Project Tab

This tab of the InstallAware action specifies the installer project to be compiled and other options.

**Project filename:** The filename of the project to compile (required).

**Build type:** Specifies the type of build output to generate.

**Use new revision code for build:** If checked, a new revision code will be used for this build.

**Compiler variables:** Defines compiler variable-value pairs for conditional compilation. Enter a definition variable name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.6.7.2 Optimizations Tab

This tab of the InstallAware action specifies optimizations (requires InstallAware v8 or later).

**MSI file component optimization:** Enables or disables this option (runtime performance).

**MSI registry component optimization:** Enables or disables this option (runtime performance).

**Unstable MSIcode statement ID generation:** Enables or disables this option (build performance).

**Partially stable MSIcode statement ID generation:** Enables or disables this option (blend).

### 3.6.7.3 Options Tab

This tab of the InstallAware action specifies additional options.

**Default language:** Specifies the default installer language to be used for this build.

**Password:** Specifies the 256-bit AES encryption password to use for the installation file (optional).

**Output folder:** Specifies a custom build output folder (optional).

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to the InstallAware command-line compiler (optional).

**Compiler executable filename:** Use this field to specify the InstallAware compiler executable if the action is unable to locate it or multiple versions are installed (optional). If not specified, the executable is located automatically.

### 3.6.8 Installer2Go / witem Installer

This action creates a step to compile an Installer2Go / witem Installer project file. From the inputs, the witem Installer / Installer2Go command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with Installer2Go and witem Installer version 4 and may also work with other versions.

**Project filename:** Specifies the project file to compile (required).

**Output folder:** Specifies the build output folder (optional). If not specified, the output folder from the project is used.

**Property names/values:** Defines property name-value pairs in the project file (optional). Enter a variable name and value in the edit fields, and click *Insert* to add to the list. Select an item in the list to update its value or delete it from the list.

**Additional command-line options:** This option can be used to specify additional flags to pass to the Installer2Go / witem Installer compiler. See *Command Line Builder* in the Installer2Go / witem Installer help index for the available options.

**Override compiler executable filename:** If this field is empty, the action automatically locates the Installer2Go / witem Installer command-line executable. This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.9 InstallShield

This action creates a step to compile an [InstallShield](#) Express, Premier, Professional, Developer, DevStudio, or PackageForTheWeb setup script. The project filename can be entered or browsed to, the type of compiler is selected, and the executable can be located automatically or manually. From these inputs, the InstallShield command-line call is generated and run when the step is built.

#### Project tab

#### Parameters tab

#### Options tab

#### Options (More) tab

#### MSI tab

#### Advanced Tab

#### Remote Tab

This action has been tested with the following product versions and may also work with other versions:

Product	Versions Tested
Developer	7.02, 8.0
DevStudio	9.0
Express	3.53, 4.0, 5.0 SP2, X, 12, 2008 thru 2015
Package4Web	4.00
Premier	11, 11.5, 12, 2008 thru 2015
Professional	7.0, X, 10 thru 12, 2008 thru 2015

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.9.1 Project Tab

This tab of the InstallShield action specifies information about the project being compiled.

**Filename:** The filename of the project to compile (required).

**InstallShield compiler to call:** Identifies which InstallShield product is being used (determines the default compiler executable location and filename and base command-line flags). Select the *All v2008-2014*, *DevStudio v9-12*, *Express v12+* option for all Professional, Premier, and DevStudio versions 9 or later and all Express versions 12 or later.

*Note:* InstallShield Professional versions prior to 7.0 are not supported by this action (InstallShield Pro uses a two-step build process [Compile.exe, then IsBuild.exe] but the executables are stored in different locations in version 6.x). For earlier versions, generate the batch file within InstallShield (from the Build menu) and run it from a Run Program action with a command of %DOSCMD% call "<path>\Build.bat".

*Options that apply only to Professional, Premier, Developer, and DevStudio:*

**Configurations:** Specifies the configurations for the release (optional, one per line).

### Setting the Product Version

The following VBScript code can be used in the InstallShield step's vblld\_StepStarted script event to set or increment the product version in an InstallShield Developer, DevStudio, X, or later project:

```
' create the InstallShield Developer automation object
Set objInst = CreateObject("ISWiAuto21.ISWiProject") ' for InstallShield 2014,
including standalone (with Automation Interface installed)
' use "ISWiAuto20.ISWiProject" for InstallShield 2013, including standalone (with
Automation Interface installed)
' use "ISWiAuto19.ISWiProject" for InstallShield 2012 Spring Edition, including
standalone (with Automation Interface installed)
' use "ISWiAuto18.ISWiProject" for InstallShield 2012
' use "ISWiAuto17.ISWiProject" for InstallShield 2011
' use "ISWiAuto16.ISWiProject" for InstallShield 2010
' use "ISWiAuto15.ISWiProject" for InstallShield 2009 (use SAAuto15.ISWiProject for
standalone)
' use "ISWiAuto14.ISWiProject" for InstallShield 2008 (use SAAuto14.ISWiProject for
standalone)
' use "ISWiAuto12.ISWiProject" for InstallShield 12 (use SAAuto12.ISWiProject for
standalone)
' use "ISWiAuto1150.ISWiProject" for InstallShield 11.5 (use SAAuto1150.ISWiProject
for standalone)
```

```

' use "ISWiAuto11.ISWiProject" for InstallShield 11 (use SAAuto11.ISWiProject for
standalone)
' use "ISWiAuto1050.ISWiProject" for InstallShield 10.5
' use "ISWiAuto10.ISWiProject" for InstallShield X
' use "ISWiAutomation9.ISWiProject" for DevStudio 9
' use "ISWiAutomation.ISWiProject" for InstallShield Developer

' open the project file
objInst.OpenProject Step.ExpProperty(Builder, "Filename")

' this code demonstrates incrementing the revision version number
' retrieve and split the ProductVersion
verArr = Split(objInst.ProductVersion, ".")
' increment last field
verArr(UBound(verArr)) = CStr(verArr(UBound(verArr)) + 1)
' update the project
objInst.ProductVersion = Join(verArr, ".")

' use this code to set the version to a specific macro value
' objInst.ProductVersion = Application.ExpandMacrosAndScript("%BUILD_VER%")

' and save changes
objInst.SaveProject
objInst.CloseProject

```

*Notes:*

- InstallShield 2008 and later also support specifying the product version on the Properties tab.
- The InstallShield object model is only provided in a 32-bit version, so it can't be called from the 64-bit version of Visual Build.

### 3.6.9.2 Parameters Tab

This tab of the InstallShield action specifies additional compilation parameters.

**Prerequisites search path:** Specifies any folders that contain InstallShield prerequisite files that are referenced by your project (optional, one per line).

*Note:* The above option applies only to InstallShield v2012 and later.

**Path variable names/values:** Specifies path variable values as if it specified in the Test Value column of the Path Variables view. Enter a path variable name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Preprocessor definition names/values:** Specifies preprocessor definitions as if specified in the Product Configuration "Preprocessor Defines" property. Enter a define name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

*Note:* These options apply to InstallShield v11 and later.

### 3.6.9.3 Options Tab

This tab of the InstallShield action specifies additional options.

**Compression:** This option is specific to Windows Installer projects. It specifies whether the release is compressed into one file or remains uncompressed in multiple files. If Default is chosen for a release that already exists, the configuration is based on what was specified in the InstallShield interface. If the parameter is omitted for a new release, the files remain uncompressed.

**Create setup.exe:** For Windows Installer-based projects, this field specifies whether to create a Setup.exe along with the installation. For merge module projects, choosing *Don't create* will cause the

merge module to be built and then copied to the merge modules folder, and *Create* will cause the merge module to only be built but not copied to the merge modules folder.

**Compilation:**

- **Build tables only:** Builds only the Windows Installer tables for your release. If you have not built this installation already, a new .msi file is created, but no files are added to your installation. If you have built your installation already, the .msi file is updated when all the tables are built, but, again no files are transferred. Ideally, this option is to be used to test the user interface of your installation.
- **Build tables and refresh:** Builds the Windows Installer tables and refreshes files. This option rebuilds your .msi file and updates the Files table, including any new or changed files in your installation. Changed files are updated only if the size or time stamp differs from the copy already included in the build. References to deleted files are removed from the installation, but the file remains in the build location. This type of build can be run only after a complete build has been performed, and it works only when the media is an uncompressed network image.
- **Compile only Setup.rul/Setup.inx:** Compiles only Setup.rul and streams Setup.inx into the Binary table of the .msi package, if one was previously built.

**Do not compile setup.rul:** Check this option if you do not want Setup.rul compiled as part of the build process.

**Merge module search path:** Specifies any folders that contain merge module (.msm) files referenced by your project (optional, one per line).

*Note:* The above option applies only to InstallShield v2008 and later.

**Override InstallShield command-line executable:** Overrides the default location for the InstallShield executable (optional). If blank, the compiler is located automatically based on the product version chosen on the InstallShield tab. Use this field to specify the InstallShield StandAlone compiler or to specify the executable filename if the action is unable to locate it or multiple versions are installed. For InstallShield Professional, this should be the path+filename for `IsBuild.exe` (the same path will be used and combined with `Compile.exe` for the first step).

**Additional options for to pass to InstallShield:** Use this field to enter any additional command-line flags to be passed to the InstallShield command-line compiler (optional).

**Additional options for Compile.exe:** Use this field to enter any additional command-line flags to be passed to the `Compile.exe` command-line compiler (optional). This field is used only for InstallShield Professional, which has a two-step compilation (first `Compile.exe`, then `IsBuild.exe`). This option field specifies any additional flags for the first step.

#### 3.6.9.4 Options (More) Tab

This tab of the InstallShield action specifies additional options.

**Build location:** The fully qualified path to the folder where you want the output folders and files to be placed (optional, does not apply to PackageForTheWeb).

**Media name:** Media name to compile (applies only for InstallShield Professional scripts).

**Release:** The release name as specified in the Release Wizard (optional, does not apply to PackageForTheWeb).

*Options that apply only to Professional, Premier, Developer, and DevStudio:*

**Release flags:** Specifies any release flags that you would like to include in your release. Separate multiple flags with commas (optional).

**Patch config:** If provided, builds a standard patch specified in the InstallShield Patch Design view (optional).

*Note:* The above option applies only to InstallShield v2009 and later.

**Treat warnings as errors:** Treats warnings that occur during the build process as errors. Each warning increments the error count by one..

**Stop on error condition:** If you want the build to stop when it encounters an error, check this option. If you want the build to stop when it encounters a warning, use this parameter in conjunction with warnings option.

### 3.6.9.5 MSI Tab

This tab of the InstallShield action specifies options for Windows Installer-based projects.

*Options that apply only to InstallShield v11 and later:*

**Validation file:** The validation file to validate the built .msi package (optional, applies only to Windows Installer-based projects).

*Options that apply only to InstallShield 2008 and later:*

**Product version:** Specifies the product version to use when compiling (optional). Provides an alternative to the automation interface for updating the product version.

**Property names/values:** Overrides Windows Installer properties when compiling (optional). Enter a property name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

*Options that apply only to InstallShield 2009 and later:*

**Minimum target MSI version:** Specifies the minimum version of Windows Installer that the installation requires on the target machine (optional). The default is the latest version of Windows Installer that the InstallShield interface supports.

**Minimum target .NET version:** Specifies the minimum version of the .NET Framework that the installation requires on the target machine (optional). The default is the latest version of the .NET Framework that the InstallShield interface supports.

**.NET Framework path:** Specifies the path to the Microsoft .NET Framework on the build machine (optional).

**Skip upgrade validators:** Skips the upgrade validators at the end of the build.

### 3.6.10 MSISStudio

This action creates a step to compile an MSISStudio project file. From the inputs, the MSISStudio command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with version 4 of MSISStudio and may also work with other versions.

**Project filename:** Specifies the MSISStudio project file to compile (required).

**Release name:** Specifies the release name to compile (optional).

**Additional command-line options:** This option can be used to specify additional flags to pass to the



MSIStudio compiler. See *Command line* in the MSIStudio help index for the available options.

**Override compiler executable filename:** If this field is empty, the action automatically locates the MSIStudio command-line executable by finding the executable associated with the MSIStudio project filename extension (.msxp). This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.11 NSIS

This action creates a step to compile an [NSIS](#) script.

#### Script tab

#### Options tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- This action has been tested with versions 2 and 3 of NSIS and may also work with other versions.
- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.11.1 Script Tab

This tab of the NSIS action specifies information about the script being compiled.

**Script filename:** The filename of the script to compile (required).

**Log level:** Specifies the amount of logging to output.

**Log information on options used to compile:** Logs information on what options were used to compile makensis was compiled with.

**Don't use defaults:** Disables inclusion of <path to makensis.exe>\nsisconf.nsh. If unchecked, installer defaults are set from nsisconf.nsh.

**Don't change current directory:** Disables the current directory change to that of the .nsi file.

**Log filename:** Specifies the log file to write output to (optional). Writes and reads from stdout if blank.

**Symbol definitions:** Add symbols to the globally defined list (See !define). Enter a definition name and (optionally) its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.6.11.2 Options Tab

This tab of the NSIS action specifies additional options.

**Code to execute:** Execute the code you specify. Enter the code in the edit and click Insert to add to the list. Select an item in the list and click Delete to remove it from the list.

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to the NSIS command-line compiler (optional).

**NSIS compiler executable filename:** Use this field to specify the NSIS compiler executable if the action is unable to locate it or multiple versions are installed (optional). If not specified, the executable is located automatically.

### 3.6.12 SetupBuilder

This action creates a step to compile a SetupBuilder setup project.

#### Project tab

#### Variables tab

#### Options tab

#### Advanced Tab

#### Remote Tab

This action has been tested with versions 6 thru 10 of SetupBuilder and may also work with other versions.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.12.1 Project Tab

This tab of the SetupBuilder action specifies the installer project to be compiled and other options.

**Project filename:** The filename of the project to compile (required).

**Release:** The release to build (optional).

**Symbols to define:** Specifies symbol definition name-value pairs to set (optional). Enter a definition name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.6.12.2 Variables Tab

This tab of the SetupBuilder action specifies variables to set.

**Variables to set:** Defines variable name-value pairs to set (optional). Enter a variable name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.6.12.3 Options Tab

This tab of the SetupBuilder action specifies additional options.

**Capture log output:** If checked, enables and logs the SetupBuilder log file contents to the build output (requires SetupBuilder 6.7 or later).

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to the ExpertInstall or QuickInstall command-line compiler (optional).

**Compiler executable filename:** Use this field to specify the ExpertInstall or QuickInstall compiler executable if the action is unable to locate it or multiple versions are installed (optional). If not specified, the executable is located automatically.

**Constants to set:** Defines constant name-value pairs to set (optional). Enter a constant name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.6.13 Setup Factory / MSI Factory

This action creates a step to compile an Indigo Rose Setup Factory, MSI Factory, or Setup Factory for Windows Installer project file. From the inputs, the Setup Factory command-line is constructed and called when the action is built, and any log output is captured to the log view and file.

This action has been tested with *Setup Factory* versions 7 thru 9, *MSI Factory* version 2, and *Setup Factory for Windows Installer* version 1 and may also work with other versions.

**Project filename:** Specifies the Setup Factory project file to compile (required).

**Build configuration:** Specifies the build configuration to use in the build (optional). If not specified, the first build configuration found in the project is used. Does not apply for Setup Factory for Windows Installer.

**INI file with design-time constants:** Lets you specify an INI file that contains design-time constants to override the ones in the project. You can define as many design-time constants as you want in the INI file, with each constant on a separate line beneath the [Constants] section. Each constant that is defined in the INI file must already be defined in the project file. For example:

```
[Constants]
#OUTPUTDIR#=C:\Output\Foobar 2002\Release
#SETUPNAME#=foobar2002setup.exe
#BUILD#=release
```

**Additional command-line options:** This option can be used to specify additional flags to pass to the Setup Factory compiler. See *Command line* in the Setup Factory help index for the available options.

**Override compiler executable filename:** If this field is empty, the action automatically locates the Setup Factory command-line executable by finding the executable associated with the Setup Factory project filename extension. This can be overridden by specifying the executable filename here.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.6.14 Windows Installer

This action creates a step to install, repair, or uninstall a Windows Installer (.MSI) file. From the inputs, the msiexec command-line call is generated and run when the step is built.

**Input Tab**

**Repair Tab**

**Options Tab**

**Advanced Tab**

## Remote Tab

### 3.6.14.1 Input Tab

This tab of the Windows Installer action specifies information about the file being installed.

**Filename:** The Windows Installer package file or product code (GUID) to install, repair, or uninstall (required).

**Operation:** The operation to perform.

**User interface level:** Specifies the level of user interface that will be displayed by the installer. Choose *None* for a completely automated install.

**Restart options:** Specifies a restart option after installation is complete.

**Property names/values:** Defines property name-value pairs to set (optional). Enter a property name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.6.14.2 Repair Tab

This tab of the Windows Installer action specifies repair options (used only for a Repair operation).

**Repair files option:** Specifies repair options for files.

**Repair user-specific registry entries:** Determines whether user registry entries are repaired.

**Repair machine-specific registry entries:** Determines whether machine registry entries are repaired.

**Overwrite shortcuts:** Determines whether shortcuts are overwritten.

**Run from source:** If checked, runs from source and recaches local package.

### 3.6.14.3 Options Tab

This tab of the Windows Installer action specifies additional options.

**Log filename:** Specifies a log filename to write log output to.

**Log all information:** If checked, all logging information except verbose and debugging output, is generated.

**Log verbose output:** If checked, verbose output is logged.

**Log debugging information:** If checked, extra debugging information is logged.

**Append to existing file:** If checked, the log file is appended to if it already exists.

**Flush each line:** If checked, each line is flushed to the log file as it is written.

**Additional command-line options:** Use this field to enter any additional command-line flags to be passed to msiexec (optional).

### 3.6.15 Wise Setup

This action creates a step to compile a Wise Installation System, InstallMaster, Wise for Windows Installer, Wise Installation Studio, or Wise Package Studio setup script. The appropriate Wise command-line will be constructed and executed when the step is built.

#### Wise tab

#### Options tab

#### Advanced Tab

#### Remote Tab

This action has been tested with the following product versions and may also work with other versions:

Product	Versions Tested
Wise for Windows Installer	4.2, 5.1, 5.21, 6.0, 6.2 (Professional and Enterprise versions)
Wise Installation Studio	7.0
Wise Installation System	9.0
Wise Package Studio	7.0
InstallMaster	7.04

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.6.15.1 Wise Tab

This tab of the Wise action specifies information about the script being compiled.

**Filename:** The filename of the script to compile.

**Property names/values:** Set property or compiler variable values to be passed to the install script. Enter a property name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

*Fields that apply to Wise for Windows Installer:*

**Release:** Compile only the specified release from an installation containing multiple releases (optional).

**Output file:** Sets output file to the specified path+filename (must be an absolute path).

*Note:* For Wise for Windows Installer, the following VBScript code can be used in the Wise step's *vblid\_StepStarted* script event to update values in tables of the script:

```
' create the Wise automation object
Set wise = CreateObject("WFWI.Document")

' open the install script
fname = Step.ExpProperty(Builder, "Filename")
wise.Open fname

' update a value in the Summary table
Set tblSumm = wise.WTables("Summary")
tblSumm.WRows.Row("Comments").WColumns("Value").Data = Application.
```

```
ExpandMacrosAndScript( "%PROJ_COMMENT%" )  
  
' save changes to the script  
wise.Save fname
```

### 3.6.15.2 Options Tab

This tab of the Wise action specifies additional options.

**Do not run silently:** Doesn't add /s command-line flag if checked.

**Automatically locate the executable:** Determines if the compiler is located automatically or manually. If checked, the action automatically locates the command-line executable to use by finding the executable associated with the extension of the file entered in the Filename field. If unchecked, the full and filename to the compiler executable must be entered in the following field. Use if multiple versions of the compiler are installed or to specify the executable file if the action is unable to locate it.

**Additional options:** Use this field to enter any additional command-line flags to be passed to the command-line compiler. See the Wise product documentation for available flags.

*Note:* To use the Wise Clean Build functionality, use a Run Program step with a command of `c:\Path\To\wisebuild.exe`.

### 3.6.16 WiX

This action creates a step to perform an operation on a Windows Installer XML (WiX) project. From the inputs, the WiX command-line tools are run when the step is built.

#### Project Tab

#### Compiler Tab

#### Linker Tab

#### Linker (more) Tab

#### Decompiler Tab

#### Harvester Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with WiX 2 thru 4 and may also work with other versions. Some options require v3 or later, and v3.0.1821 or later is required for the Harvest operation.

*Notes:*

- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- The Make VS.NET action can be used to build .wixproj project files and Visual Studio solutions containing WiX/Votive projects.

### 3.6.16.1 Project Tab

This tab of the WiX action specifies common WiX settings.

**Input file:** The WiX input file to process (required, .wxs for compile or compile+link operation; .wixobj for link operation; .msi for decompile operation; directory, filename, or web site for harvest operation).

**Operation:** The operation to perform. Compile calls the WiX compiler (candle.exe); Link calls the linker (light.exe); Decompile calls the decompiler (dark.exe); and Harvest calls the harvester (heat.exe).

**Output level:** Specifies the level of logging.

**Pedantic checks:** Specifies pedantic checking to perform (optional).

**Warning level:** Specifies the warning level.

**Treat warnings as errors:** Specifies whether warnings will fail the step.

**Suppress output of logo information:** Self-explanatory.

**Suppress schema validation of documents:** If checked, no validation of documents is performed.

**Extension assembly:** Specifies an extension assembly or class, assembly (optional).

**WiX path:** Specifies the path containing the WiX compiler, linker, decompiler, and harvester command-line tools (optional). If not specified, the action will attempt to locate WiX automatically.

### 3.6.16.2 Compiler Tab

This tab of the WiX action specifies compiler options.

**Output filename:** Specifies the WiX object filename (optional, uses input filename and .wixobj extension if not specified).

**Only do validation of documents:** Only validates documents if checked and does not compile or link.

**Show source trace for errors, warnings, and verbose messages:** Self-explanatory.

**Preprocessor parameters/values to set:** Defines preprocessor parameter name-value pairs to set (optional). Enter a parameter name and value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Additional compiler command-line options:** Specifies additional compiler (candle.exe) command-line options (optional). Type candle<Enter> at a Command Prompt to see the available options.

### 3.6.16.3 Linker Tab

This tab of the WiX action specifies linker options.

**Output filename:** Specifies the output filename (optional, uses the input filename and .msi or .xml extension if not specified).

**Output XML instead of MSI format:** If checked, generates XML output instead of MSI.

**Base path:** Specifies a base path to generate files in (optional).

**Output path:** Specifies a base output path for uncompressed images (optional).

**Cache path:** Specifies a path to cache cabinet files in (optional).

**Additional command-line options:** Use this field to enter any additional linker (light.exe) command-line flags. Type light<Enter> at a Command Prompt to see the available options.

#### 3.6.16.4 Linker (more) Tab

This tab of the WiX action specifies additional linker options.

**Reuse cabinets from cache:** Reuses cabinets from cache if available instead of regenerating.

**Do not delete temporary files:** If checked, temporary files that are generated will not be deleted.

**Suppress default admin sequence actions:** Self-explanatory.

**Suppress default UI sequence actions:** Self-explanatory.

**Suppress intermediate file version mismatch checking:** Self-explanatory.

**Suppress dropping unreal tables to output image:** Self-explanatory.

**Suppress processing data in MsiAssembly table:** Self-explanatory.

**Suppress file info:** Self-explanatory.

**Treat identical rows as warnings:** Self-explanatory.

**Do not get assembly name info for assemblies:** Self-explanatory.

**Tag sectionId attributes in tuples:** Self-explanatory.

#### 3.6.16.5 Decompiler Tab

This tab of the WiX action specifies decompiler options.

**Output filename:** The WiX source file to decompile to (optional, defaults to name of MSI file from the Input file field on the Project tab, with .wxs extension).

**Do not delete temporary files:** If checked, temporary files that are generated will not be deleted.

**Suppress dropping empty tables:** Prevents dropping of empty tables.

**Suppress relative action sequencing:** Prevents generation of relative action sequencing.

**Suppress decompiling UI-related tables:** Prevents generation of UI-related tables.

**Additional command-line options:** Use this field to enter any additional decompiler (dark.exe) command-line flags. Type dark<Enter> at a Command Prompt to see the available options.

#### 3.6.16.6 Harvester Tab

This tab of the WiX action specifies harvester options.

**Harvest type:** Specifies the type of data to harvest (a directory, filename, or web site).

**Output filename:** The WiX source file to generate (optional, defaults to folder of the input file with



.wxs extension).

**Additional command-line options:** Use this field to enter any additional harvester (heat.exe) command-line flags. Type heat<Enter> at a Command Prompt to see the available options.

## 3.7 Localization

### 3.7.1 Multilizer

This action builds a Multilizer project. The command-line application (`MlBuild.exe`) is invoked. The action is used to configure the options available for the command-line version of Multilizer (available in the Enterprise edition only).

#### Input Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- This action has been tested with Multilizer versions 6 thru 11 and may work with other versions as well.

#### 3.7.1.1 Input Tab

This tab of the Multilizer action configures input settings.

**Project file:** Specifies a `.mpr` or `.m7p` project file to use (required).

**Task:** The tasks to perform (required, separate multiple with spaces).

**Language codes:** Specifies language codes to include for the create, duplicate, export, import, and translate tasks (optional, one per line). Code format is `ll[-CC]` where `ll` is a two-character ISO-639 language code and `CC` is a two-character ISO-3166 country code.

**Source type:** Specifies the source type for add and import commands (optional).

#### 3.7.1.2 Options Tab

This tab of the Multilizer action configures additional options.

**Quiet mode:** Only errors will be logged.

**Additional options:** Used to specify any additional MlBuild command-line options (optional). See the Multilizer documentation for available options.

**Multilizer executable:** Overrides the default location of `MlBuild.exe` (optional). Useful if multiple versions of Multilizer are installed.

### 3.7.2 Sisulizer

This action builds a Sisulizer project. The command-line application (`SlMake.exe`) is invoked. The action is used to configure the options available for the command-line version of Sisulizer (available in the Enterprise edition only).

#### Input Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- This action has been tested with Sisulizer versions 1, 2 (2008), 2010, 3, and 4 and may work with other versions as well.

#### 3.7.2.1 Input Tab

This tab of the Sisulizer action configures input settings.

**Project file:** Specifies a `.slp` project file to use (required).

**Task:** The tasks to perform (required, separate multiple with spaces).

**Language codes:** Specifies language codes to include for the create, duplicate, export, import, and translate tasks (optional, one per line). Code format is `ll[CC]` where `ll` is a two-character ISO-639 language code and `CC` is a two-character ISO-3166 country code.

**Source type:** Specifies the source type for add and import commands (optional).

#### 3.7.2.2 Options Tab

This tab of the Sisulizer action configures additional options.

**Names of sources to include:** One or more strings (separated by semicolons) with names of sources to include in the build (optional).

**Names of sources to exclude:** One or more strings (separated by semicolons) with names of sources to exclude from the build (optional).

**Quiet mode:** Only errors will be logged.

**Additional options:** Used to specify any additional `SlMake` command-line options (optional). See the Sisulizer documentation for available options.

**Sisulizer executable:** Overrides the default location of `SlMake.exe` (optional). Useful if multiple versions of Sisulizer are installed.

## 3.8 Microsoft

Visual Build provides built-in support for creating automated builds for many Microsoft development tools. See the following topics for more details:

- MSBuild
- SourceSafe
- Team Build
- Team Foundation
- Team Test
- Visual Basic 6
- Visual C++ 6
- Visual Studio .NET and Embarcadero Delphi Prism
- VS.NET Get Version
- VS6 Get Version
- Visual Studio and .NET SDK Macros
- Visual Studio Integration

Several samples of automated builds using these tools are also provided.

### 3.8.1 Make VB6

The Make VB6 custom action creates a step for configuring all the available options for building Microsoft Visual Basic 6.0 projects and groups. This action provides functionality that is missing from Microsoft's VB compiler. It adds the ability to process large group files, 'make' intelligence to prevent unnecessary builds and speed up the build process, setting project compatibility and symbolic debug flags, setting project DLL base addresses, setting or incrementing project versions, and more. See the VStudio.bld sample for sample usage.

When the step is built, for project group (VBG) files, this action parses the group file for individual projects and processes each one. It builds a list of the VBP files in the group, their target (DLL/EXE) filenames, and each project's references to other projects in the group (it recognizes references to VBPs and executables). It then iterates over the list of projects, processing projects in the correct order according to their references. If a project is set to *project compatibility* or *no compatibility* and other projects in the group reference that project, Visual Build detects the type lib GUID/version (reference) change and updates all references to the project from other projects in the group. This prevents broken references, compile errors, and upgrade reminders on OCX references that can interfere with automated builds.

For individual VBP files, the action analyzes the project, and if the target executable doesn't exist or is older than any source files in the project, it invokes the VB compiler to build the project. It logs the filename as it is being processed and a status indicating what is being done for that project (setting compatibility, incrementing versions, building, setting compatibility, etc.). The action detects if the specified changes have already been made and only modifies the project files if necessary.

#### Project/Group Tab

#### Compatibility Tab

#### Versions Tab

#### Properties Tab

#### Options Tab

## Advanced Tab

### Remote Tab

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.8.1.1 Project/Group Tab

This tab of the Make VB6 action specifies information about the project or group being built.

**Filename:** The Visual Basic project or project group file to build. It supports VBP, VBG, EBP, and EBG files and processes all projects in group files. Visual Basic sometimes has problems with large group (VBG) files, but there are no limitations on VBG size when building with this action, since the VB compiler is invoked for each individual VBP in the group.

When building Visual Basic project group (VBG) files, this action parses the group file for individual projects and processes each one. It builds a list of the VBP files in the group, their target (DLL/EXE) filenames, and each project's references to other projects in the group (it recognizes references to VBPs and executables). It then iterates over the list of projects, processing projects in the correct order according to their references. If a project is set to *project compatibility* or *no compatibility* and other projects in the group reference that project, Visual Build detects the type lib GUID/version (reference) change and updates all references to the project from other projects in the group. This prevents broken references, compile errors, and upgrade reminders on OCX references that can interfere with automated builds. For individual VBP files, the action analyzes the project, and if the target executable doesn't exist or is older than any source files in the project, it invokes the VB compiler to build the project.

*Note:* The Make VB6 Action requires that all VBP filenames within a single VBG be unique. If multiple projects in the group have the same filename, rename the VBP files so that they are unique, and update the VBG with the changes as well. The project names stored in the VBP files do not need to be changed (but be aware that it is dangerous to have multiple VB COM projects with the same project name since this can result in ProgID clashes when registered).

**Don't build:** Does not build any projects, but updates other setting such as versions and project properties.

**Force a build of all projects:** If unchecked, only projects whose executable is older than one of the project source files are built. The action analyzes all the project dependencies, and only builds a project if the target executable is older than any of the files in the project or its references. Unchecking this option will speed up your builds by preventing unnecessary recompilation and linking. If checked, all projects are rebuilt, regardless of the target date.

**Exclude references:** Exclude references from consideration when performing incremental builds. Normally, the timestamp of all project references are also compared to a project's executable to determine if a project needs to be built. Using this flag will cause references to be excluded from consideration.

**Exclude project file:** Excludes the project file from consideration when performing incremental builds. If unchecked, the timestamp of the project file is compared to a project's executable to determine if a project needs to be built. Checking this flag will cause the project file to be excluded from consideration.

**Show reason for incremental build:** If checked and a project needs to be built, the reason for building it will be logged (either the target did not exist or the filename of the file that was newer than the target).

**Create symbolic debug info:** Toggle inclusion of symbolic debug info when building.

**Clear the target executable:** Clear target executable's read-only attribute before building.

**Continue building non-dependent projects on build failure:** Continue building any non-dependent projects if a project in a workspace or group fails to build.

**Log a list of failed projects:** If checked, if any projects fail to build, the project filenames will be logged at the end of the step's build output.

**Restore read-only attribute of any files modified by action:** If checked, any project files modified for the build (to set/increment versions, properties, etc.) which are marked read-only before modification will be set back to read-only after being modified.

**Treat circular references within project group as an error:** If checked and a circular reference is detected while building a project group, the step will fail without attempting to build the project with the circular reference.

**Log references found within project group:** If checked, when building a project group, each project's references to other projects in the group will be logged.

### 3.8.1.2 Compatibility Tab

This tab of the Make VB6 action configures compatibility and output path settings for the build.

**Set version compatibility before building:** This option applies only to ActiveX DLL, EXE, and Control project types. If set to Binary, Project, or No compatibility, before a project is built, the project file is updated to the specified compatibility setting. This is a useful in preventing missing references or compile errors if compatibility has been broken on VB projects.

When binary compatibility is set, interface and class GUIDs are not modified (but compile errors will occur if any interface signatures were changed since the previous build). When project compatibility is set, the interface and class GUIDs are regenerated, but the type lib GUID (what the reference is set to) is not changed. When no compatibility is set, all GUIDs are regenerated. When specifying a project group (.vbg) file on the Project/Group tab, any references to the project in the group will be fixed up after building with Project or No compatibility. When performing incremental builds, this action looks at all references and builds the project if any referenced files are newer than that project's executable file.

**Update compatibility directory:** Set the compatibility path before building.

**Set binary compatibility:** This option applies only to ActiveX DLL, EXE, and Control project types. If checked, after a project is processed (whether or not it needed to be built), the project file is updated and Binary Compatibility is set to the project's target executable. This is a great convenience for developers, since remembering to do this for each project after the first build is easy to forget. This eliminates a lot of *Error 429 – ActiveX component can't create object* errors and keeps the registry from getting cluttered.

**Update the target executable directory:** Set the target output path before building.

### 3.8.1.3 Versions Tab

This tab of the Make VB6 action configures versioning and base address info to be applied.

**Increment Version:** The increment version option will increment the project version only if the project is built. This ensures that your installs work correctly and copy updated files when comparing the file version. If a project needs to be rebuilt (Force build was selected or the target is out of date), the action will increment the Revision Version Number in the project file before building. The VB compiler

stores the project version number in the FileVersion and ProductVersion fields of the version info.

**Set Version:** The set version option will set the project version to the specified value. This is useful if you want your executables to be marked with a specific build number. The version should be entered in the format 9.9.9 (e.g., 8.1.12). The Major, Minor, and Revision numbers in the project file are set to the specified value before building.

**Base Address:** This action supports setting the base address of DLL and OCX projects. Normally all projects are set to the same default address, requiring most of the DLLs to be "rebased" when they are loaded into a process. By setting the base address on each project, your application will run faster since the relocation is eliminated.

If *Random value* is selected, a random address will be chosen for each project and the project file will be updated with that value (this is only done if the value is still set to its default value, so after the first time this is set it will not be changed again by Visual Build when a project is built again). Since there are over 24,000 unique base addresses available, the likelihood is very good that each component loaded into a single process will get a unique address.

If *Set to a specific value* is selected, the base address for the project will be set to that value (it must be in the range 1000000 to 7000000 hex). This option should only be used for single projects, not groups.

#### 3.8.1.4 Properties Tab

This tab of the Make VB6 action configures project properties to be set.

Used to set project properties to values that you specify. This can be useful for quickly updating all projects in a group. The following properties can be set: Comments, Company Name, File Description, Legal Copyright, Legal Trademarks, Product Name, Project Name, and Project Description. Any or all the properties can be updated or cleared. The files are only modified if the existing values differ from those specified.

#### 3.8.1.5 Options Tab

This tab of the Make VB6 action configures additional options.

**Create .bak file when updating project file:** If checked, the previous version of the project file will be renamed to .vbp.bak before updating properties, versions, base address, target directory, debug info, compatibility settings, or references in the project file. If this option is unchecked, no backup copy of the project file will be kept when updating.

**Do not use temporary copy of project file for building:** If unchecked, the original project file will be copied to a temporary tmp~~original.vbp file and that copy will be used to build (so that building will work even if the original project file is open in the VB6 IDE). If this option is checked, the original .vbp file will be built instead.

**Additional Command-Line Options:** This action invokes the VB command-line program to perform the builds, and this option can be used to specify additional flags to pass to the tools. For instance, the VB compiler supports a /D flag to specify conditional compilation constants. Details on the VB syntax are available [here](#). The values entered in this field are passed through directly to the command-line program.

**Override default VB compiler location:** If left blank, the action will automatically locate the VB6 command-line compiler by looking for this registry entry:  
HKEY\_CLASSES\_ROOT\VisualBasic.Project\shell\Make\command\ and extracting out the exe filename. By providing a full path and filename for this field, that executable will be used instead.

## 3.8.2 Make VC6

The Make VC6 custom action creates a step to build Microsoft Visual C++ 6.0 and Embedded Visual C++ 3.0 and 4.0 projects and workspaces and to set or increment version and/or property information in a resource file.

This custom action adds functionality that is missing from Microsoft's command-line tools--It supports building projects and multi-project workspaces without having to specify the name of each project, setting the project's DLL base address, setting or incrementing project versions and properties across workspaces, projects, or resource files, and more. See the VStudio.bld sample for sample usage.

When building projects, this action analyzes the project or workspace and all dependencies and invokes the MSDEV or EVC command-line executable for each project in the correct order. All build output from the compiler, linker, etc. is displayed. When building workspaces, the action adds /NORECURSE to the command-line when calling MSDEV since the action analyzes the dependencies (without this flag, projects in a multi-project workspace would be built multiple times, since both the action and MSDEV would analyze the project dependencies).

If any errors occur while building, the error output is logged to the Output pane, and the action aborts. After correcting the errors, build the project again and it will continue from the point of failure, since all successfully completed projects will be up-to-date and will not be built again (unless the Force flag is used).

### Project/Workspace Tab

### Versions Tab

### Properties Tab

### Options Tab

### Advanced Tab

### Remote Tab

*Notes:*

- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- This action can also be used to build VC6 projects and workspaces with Xoreax IncrediBuild (v2.0 and later). On the Options tab, set the *Override default MSDEV location* (or define and use a global macro) to the full path to `BuildConsole.exe` (usually `C:\Program Files\Xoreax\IncrediBuild\BuildConsole.exe`), and enter any IncrediBuild-specific switches in the *Additional Command-line Options* field.

### 3.8.2.1 Project/Workspace Tab

This tab of the Make VC6 action configures information about the project or workspace being processed.

**Filename:** The Visual C++ project or workspace file to build or resource file to update. DSP, DSW, VCP, VCW, and RC files are supported. Visual C++ provides no simple way to specify a single configuration for all projects in a workspace (DSW) when building from the command-line, but this action also overcomes this limitation.

*Note:* When updating individual RC files, only the inputs from the Versions and Properties tabs are

applicable.

**Don't build:** Does not build any projects, but updates other setting such as versions and project properties.

**Force a build of all projects:** Unchecking this option will speed up your builds by preventing unnecessary recompilation and linking. If checked, the /REBUILD flag is added when calling the MSDEV compiler to force a rebuild of all projects.

*Note:* To perform a *clean* operation, add /CLEAN to the additional command-line options field on the Options tab.

**Clear the target executable:** Clear target executable's read-only attribute before building.

**Continue building non-dependent projects on build failure:** Continue building any non-dependent projects if a project in a workspace fails to build.

**Log a list of failed projects:** If checked, if any projects fail to build, the project filenames will be logged at the end of the step's build output.

**Support library dependencies when building multi-project workspaces:** If checked, each project is built by specifying the workspace filename (DSW), enabling VC to automatically link with static library dependencies in the workspace. If unchecked, the DSP filename is specified for each project that is built. Building via the workspace file can be much slower, since VC must load all dependency information in the workspace once for each project that is built, so use this option only if you need this functionality (another alternative is to manually add the .LIB filename of static library dependency projects to each dependent project's link settings).

**Configuration:** Allows all configurations to be built, or just a single configuration in all projects. Visual C++ provides no simple way to specify a single configuration for all projects in a workspace (DSW) when building from the command-line, but this action also overcomes this limitation. When specifying a configuration name, enter the full configuration name, including the 'Win32 ' prefix to specify an individual configuration. Partial matching of multiple configuration names can also be performed by entering a partial configuration name (i.e., 'Release' to match 'Win32 Release MinDependency' and 'Win32 Release MinSize').

**Update the target executable directory:** Set the target output path before building

**Restore read-only attribute of any files modified by action:** If checked, any project files modified for the build (to set/increment versions, properties, etc.) which are marked read-only before modification will be set back to read-only after being modified.

### 3.8.2.2 Versions Tab

This tab of the Make VC6 action configures version and base address information to be set for all projects.

**Increment Version:** The increment version option will increment the project version only if the project is built. This ensures that your installs work correctly and copy updated files when comparing the file version. After building or rebuilding a project, the action compares the original timestamp of the target executable with the timestamp after building. If it has changed, the version in the project's RC file(s) is incremented and the project is built again (the second build will just recompile the resources and link). The FileVersion and/or ProductVersion fields (both the raw and string values) are incremented. Normally, the 4th value (sometimes called the Build number) is incremented, but the 3rd value (sometimes called the Revision number) can be incremented instead by checking that checkbox.

**Set Version:** The set version option will set the project version to the specified value. This is useful if you want your executables to be marked with a specific build number. The version should be entered



in the format 9.9.9.9 (e.g., 1.0.1.12). The version in the project's RC file(s) is updated before building. The raw version will be stored in the required "9, 9, 9, 9" format, while the string version will be stored exactly as entered.

When using the Increment or Set Version radio button, either one or both of the File or Product version checkboxes should be checked to indicate which version(s) to increment or set.

*Note:* To set a different value for the two version fields, create two Make VC6 Action steps for a single project/workspace, with the first having its *Don't build* field checked; then set the file version field in one step and the product version in the other. These steps could be placed in a subroutine and called multiple times if this functionality was needed for multiple projects or workspaces.

**Base Address:** This action supports setting the base address of DLL and OCX projects. Normally all projects are set to the same default address, requiring most of the DLLs to be "rebased" when they are loaded into a process. By setting the base address on each project, your application will run faster since the relocation is eliminated.

If *Random value* is selected, a random address will be chosen for each project and the project file will be updated with that value (this is only done if the value is still set to its default value, so after the first time this is set it will not be changed again when a project is built again). Since there are over 24,000 unique base addresses available, the likelihood is very good that each component loaded into a single process will get a unique address.

If *Set to a specific value* is selected, the base address for the project will be set to that value (it must be in the range 1000000 to 7000000 hex). This option should only be used for single projects, not workspaces. This option updates or creates the /base option in the project's link step.

### 3.8.2.3 Properties Tab

This tab of the Make VC6 action configures project properties to be assigned to all projects.

Used to set project properties in the resource file's VERSIONINFO resource to values that you specify. This can be useful for quickly updating all projects in a workspace. The following properties can be set: Comments, Company Name, File Description, Legal Copyright, Legal Trademarks, Product Name, Private Build, and Special Build. Any or all the properties can be updated or cleared. The files are only modified if the existing values differ from those specified.

### 3.8.2.4 Options Tab

This tab of the Make VC6 action configures additional options.

**Log dependencies found in workspace:** If checked, when building a multi-project workspace, each project's dependencies on other projects in the workspace will be logged.

**Additional command-line options:** This action invokes the MSDEV command-line program to perform the builds, and this option can be used to specify additional flags to pass to the tools. For instance, MSDEV supports a /USEENV flag to ignore the Tools|Options|Directories settings. Details on the MSDEV flags are covered [here](#). The values entered in this field are passed through directly to the command-line program.

**Override default MSDEV location:** If blank, the command-line MSDEV compiler is automatically located by looking for the registry entry  
HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\App  
Paths\msdev.exe\ and extracting out the exe filename. For Embedded Visual C++, the registry entries HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion\App  
Paths\evc4.exe\ (and evc.exe\) are searched and used if found. By providing a full path and filename for this field, the command-line executable specified will be used rather than looking it up.

To build VC6 projects and workspaces with Xoreax IncrediBuild (v2.0 and later), set enter the full path to `BuildConsole.exe` (usually `C:\Program Files\Xoreax\IncrediBuild\BuildConsole.exe`), and enter any IncrediBuild-specific switches in the *Additional command-line options* field.

### 3.8.3 Make VS.NET

The Make VS.NET / Make VS 2005 thru 2015 / Make Delphi Prism custom actions create a step for building Microsoft Visual Studio .NET 2002 thru 2015 and Embarcadero/CodeGear Delphi Prism projects and solutions to set or increment version and/or property information in an assembly attribute or resource file.

This action adds functionality missing from Microsoft's command-line tools: It supports setting or single-incrementing project, assembly attribute, and/or resource versions and properties, setting the output file path and attributes, building multiple or all solution configurations, upgrading solutions, setting the project's DLL base address, and more.

This action wraps the `devenv.com/msbuild.exe/vcbuild.exe` command-line compiler, modifying project settings if specified, updating resource and assembly info files as necessary, and then invoking `devenv.com` or `msbuild.exe` to perform the build. See the `VStudio.bld` sample for sample usage.

*Note:* If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### Project/Solution Tab

#### Versions Tab

#### Properties Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

#### Action Details

This action supports solutions and individual VC++.NET (managed, unmanaged, and Intel C++ Compiler projects), C#, VB.NET, LightSwitch, J# .NET, Delphi Prism (.oxygene), Web Deployment (.wdproj), Reporting Services (.rptproj), Analysis Services (.dwproj), Integration Services (.dtproj), and WiX projects (.wixproj), and all Windows, web, and Smart Device (.NET Compact Framework) project types (other project types in a solution will be built, but the added features will not be performed for them), and also individual RC and assembly attribute files. It has been tested with Visual Studio .NET 2002 thru 2015 and Delphi Prism and may also work with other versions.

The Intel C++ and Fortran compilers (tested with versions 9, 10, and 11) are also supported if:

- 1) During installation of the Intel compilers, the Visual Studio Integration option is included and all update environment variable options are checked.
- 2) The projects are converted to use the Intel C++ Project System (in the VS IDE).
- 3) The project or solution is built with `devenv` (specified in the *Override* field of the Options tab).

*Notes:*

- When using this action to build projects or solutions for Visual Studio 2005 or later, MSBuild will be used by default. Only the .NET Framework must be installed; Visual Studio does not need to be installed on the build box. This behavior can be overridden by specifying the full path+filename of the compiler executable on the Options tab. When building projects with devenv.com or VCBuild, Visual Studio or VCBuild must also be installed on the build box.
- When building Visual Studio 2002 or 2003 projects or solutions, the appropriate version of Visual Studio must be installed on the build box.
- MSBuild does not support building of setup and deployment projects (or solutions containing them). Enter `devenv` in the Override field on the Options tab to build projects or solutions containing Visual Studio 2005+ setup projects.
- This action can also be used to build solutions and projects with Xoreax [IncrediBuild](#) (v2.0 and later). On the Options tab, set the *Override default DEVENV location* (or define and use a global macro) to the full path to `BuildConsole.exe` (usually `C:\Program Files\Xoreax\IncrediBuild\BuildConsole.exe`), and enter any IncrediBuild-specific switches in the *Additional command-line options* field.

*Note:*

- See the `VStudio.bld` sample for a project utilizing this action.
- By default, the Make VS 2005, Make VS.NET, and Make Delphi Prism actions are not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

**3.8.3.1 Project/Solution Tab**

This tab of the Make VS.NET / Delphi Prism action configures information about the project or solution to be built or the project, solution, or assembly info file to be updated.

**Filename:** The Visual Studio project or solution file to build or the assembly attribute or resource file to update. The following file extensions are supported: `.sln`, `.csproj`, `.vbproj`, `.vcproj`, `.vjsproj`, `.csdproj`, `.vbdproj`, `.vdproj`, `.oxygene`, `.wixproj`, `.rc*`, `.cs`, `.vb`, `.jsl`, `.cpp`.

*Notes:*

- When updating individual assembly attribute or RC files, only the inputs from the Versions and Properties tabs are applicable.
- To build an individual web project, specify its URL in the Filename field (i.e., `http://localhost/WebApplication1/WebApplication1.csproj`).

**Don't build:** Does not build the projects; only updates versions, base address, or other properties that are specified.

**Build behavior:** Specifies which build type to perform:

- *Build:* Builds any out of date projects
- *Rebuild:* Cleans and then builds the project or solution.
- *Clean:* Deletes all intermediate files and output directories.
- *Deploy/Publish:* Builds and then deploys or publishes a project (the project/solution specified must be a deployment project or a ClickOnce application).
- *Upgrade:* Upgrades the solution and all projects to Visual Studio 2005 thru 2015 format (applies only to Visual Studio 2005 thru 2015). When using this option, checking the *Clear the target executable* option will make the solution and all project files writeable before processing.
- *Web Publish:* Publishes the project (VS 2010+ web projects only; must build with MSBuild).
- *Package:* Packages the project (VS 2010+ web projects only; must build with MSBuild).

*Note:* VS.NET 2002/2003 contain a bug that prevents clean from doing anything for non-C++ projects,

and VS.NET 2002/2003 always rebuilds non-C++ projects even when Build is specified.

**Clear the target executable's read-only flag:** Clears the read-only flag for the output file (target executable) of each matching configuration of each project if set.

**Post-process successfully built projects on solution build failure:** If checked, when building a multi-project solution, if some projects build successfully but one or more projects fail to build, any version increment logic (configured on the Versions tab) will be performed for the successfully built projects. If unchecked, the step will not perform post-processing if any projects in the solution fail to build. This option is most useful with VC++ solutions to ensure that versions get incremented in all cases.

**Parse build output for failed projects:** Sometimes, `devenv.com` will return a success (0) exit code even if one or more projects failed to build. If this box is checked, the action also parses the build output to verify that 0 projects failed to build, and if an error condition is found, the step will also be marked as failed. This flag does not apply when building Visual Studio 2005 thru 2015 and Delphi Prism projects with `msbuild.exe`.

**Configuration:** Specifies the project or solution configuration to build. Choose All to build all configurations, enter a valid configuration name or select one from the list, or enter a regular expression to match one or more configurations (i.e., `Debug.*` to match all configurations beginning with Debug). When building a `.sln` file, you must provide a valid solution configuration; when building individual project files, enter a valid project configuration (the Configuration drop-down list is populated from the Filename provided when possible).

Visual Studio .NET adds the concept of solution configurations in addition to project configurations. Solution configurations are accessed from the properties of the solution (by right-clicking on a solution in the Solution Explorer, choosing Properties, and clicking on the Configuration Properties tree item), not the project itself. For each solution configuration, VS.NET allows you to specify which project configuration will be built and also allows you to include or exclude projects from a build (accessed from *Build | Configuration Manager* on the menu in Visual Studio .NET). These same settings will be used by Visual Build when building the solution.

**Update the output directory for each project:** If provided, the output directory for each matching configuration of each project will be updated to the specified value.

*Notes:*

- For VC++ projects, this updates the Output Directory setting (configured at *Project | Properties | Configuration Properties | General* in Visual Studio); the Output File setting (configured at *Project | Properties | Linker | General* or *Project | Properties | Librarian | General*) must reference `$(OutDir)` for this to have an effect.
- Visual Studio supports several macros that can be used here (for instance, `$(ConfigurationName)`, `$(ProjectName)`, etc.).

**Restore read-only attribute of any files modified by action:** If checked, any project files modified for the build (to set/increment versions, properties, etc.) which are marked read-only before modification will be set back to read-only after being modified.

**Include devenv /project flag when building individual projects:** If checked, an individual project file is specified in the Filename field, and `devenv` is used to build (on the Options tab), the `/project` flag will be included to specify that only that project should be built even if `devenv` finds a solution containing that project. This can be needed to prevent `devenv` from finding and building a related solution instead, but in other circumstances it can cause a build error.

### 3.8.3.2 Versions Tab

This tab of the Make VS.NET / Delphi Prism action specifies versioning and base address settings.

**Increment Version:** The increment version option will increment the project version only if the project is built (the version is always incremented when specifying an individual RC or assembly file). This ensures that your installs work correctly and copies updated files when comparing the file version. After building or rebuilding a project, the original timestamp of the target executable is compared with the timestamp after building. If it has changed, the custom action increments the version and builds the project again (the second build will just recompile the resources and link).

For managed projects containing an AssemblyInfo file (Managed C++ .NET, Visual C# and VB.NET projects) and source code files, the AssemblyFileVersion (file version), AssemblyInformationalVersion (product version), and/or AssemblyVersion attributes in the project's AssemblyInfo file are created or updated. Normally, the 4th value (sometimes called the Revision number) is incremented, but the 3rd value (sometimes called the Build number) can be incremented instead by checking that checkbox.

For Visual C++ .NET projects, the FileVersion and/or ProductVersion fields within the resource file's VERSIONINFO resource are also incremented by extracting the last number in the version and incrementing it (the format is not changed).

#### Notes:

- VS.NET defaults the Assembly version to 1.0.\* and uses the following logic when auto-incrementing: it sets the build part to the number of days since January 1st, 2000, and sets the revision part to the number of seconds since midnight, local time, divided by two. The Make VS.NET action will not change the 1.0.\* value when using its Increment option. To override VS.NET's increment logic, temporarily set the Assembly version to a specific value (i.e., 1.0.0.0) and build the step, then using the Make VS.NET action's Increment option will increment the version by a single digit (and will also allow the Get VS.NET Version action to retrieve the actual version value).
- VS.NET 2002/2003 does not implement true incremental build intelligence for C# and VB.NET projects; a project's assembly executable is always updated even if no source files were changed since the last build, which will always cause the version to be incremented for these project types when using the Increment Version option.
- The Set and Increment version capabilities are not supported and will be ignored for Visual Studio 2005 thru 2015 web projects.

**Set Version:** The set version option will set the specified version fields to the provided value. This is useful if you want your executables to be marked with a specific build number. The version should be entered in the format 9.9.9.9 (e.g., 1.0.1.12). The version(s) are updated in the resource file(s)/AssemblyInfo file before building.

For Visual C++ .NET projects containing resource files, the raw version will be stored in the required "9, 9, 9, 9" format, while the string version will be stored exactly as entered; for managed projects containing an assemblyinfo file, the version number is set to the value specified.

When using the Increment or Set Version radio button, one or more of the File, Product, and Assembly version checkboxes should be checked to indicate which version(s) to increment or set.

#### Notes:

- To set a different value for each of the version fields, create multiple Make VS.NET Action steps for a single project/solution, with all but the last having its *Don't build* field checked; then set a specific version field in each of the steps. These steps could be placed in a subroutine if different versions needed to be set on multiple projects or solutions.
- To exclude a single project in a multi-project managed solution from version set/increment, rename that project's AssemblyInfo file (i.e., AssemblyInfo.cs) in the VS.NET IDE, and the Make VS.NET action will not update versions or properties for that project.
- Only the Assembly and Product versions apply for Smart Device (Compact Framework) projects; the

File version option should be unchecked for these project types.

- For C# and VB.NET projects, if only the Assembly version is set/incremented, VS.NET will also assign that version for the file and product version fields in the executable (if a file or product version attribute is already set it will need to be removed from the projects' AssemblyInfo. files and the projects rebuilt).
- For *Setup and Deployment* projects, the ProductVersion field will be set or incremented if *Update the Product version* is checked. To update the **Product Code** with a new GUID, use a Replace in File action with *Text or regular expression to find of*

```
"ProductCode" \s*=\s*"8:([^\s]*)"
```

and Text or regular expression to replace matches with of

```
"ProductCode" = "8:[CreateObject("Scriptlet.TypeLib").Guid]"
```

**Base Address:** Normally all projects are set to the same default address, requiring most of the DLLs to be "rebased" when they are loaded into a process. By setting the base address on each project, your application will run faster since the relocation is eliminated.

If *Random value* is selected, a random address will be chosen for each project and the project file will be updated with that value (this is only done if the value is still set to its default value, so after the first time this is set it will not be changed when a project is built again). Since there are over 24,000 unique base addresses available, the likelihood is very good that each component loaded into a single process will get a unique address.

If *Set to a specific value* is selected, the base address for the project will be set to that value (it must be in the range 1000000 to 7000000 hex). This option should only be used for single project solutions.

*Notes:* The Base Address option does not apply for Smart Device (Compact Framework) or managed Visual Studio 2005 thru 2015 projects.

### 3.8.3.3 Properties Tab

This tab of the Make VS.NET / Delphi Prism action is used to configure properties to be assigned to all projects.

Used to set the assembly properties in each project's AssemblyInfo file (for managed projects) and/or the properties in each project's resource file VERSIONINFO resource (for Visual C++.NET projects) to values that you specify. This can be useful for quickly updating properties in an individual assembly or resource file, a single project, or all projects in a solution. The following properties can be set (assembly/version info name listed): Description/Comments, Company/Company Name, Title/File Description, Copyright/Legal Copyright, Trademark/Legal Trademarks, Product/Product Name, Configuration/Private Build, Culture/Special Build. Any or all the properties can be updated or cleared. The files are only modified if the existing values differ from those specified.

*Note:* The action can only set project versions and properties on web projects if the project is pointing to *localhost* as the server -- it can resolve that to a local filename for updating, but it can't update files accessible only via HTTP.

### 3.8.3.4 Options Tab

This tab of the Make VS.NET / Delphi Prism action configures additional options.

**MSBuild logging level:** Specifies the MSBuild logging level for build output. This option applies only when building Visual Studio 2005 thru 2015 and Delphi Prism projects or solutions with MSBuild.

*Note:* For Visual C++ 8.0 through 14.0 (VS2005 thru 2015) projects and solutions, full build output can be obtained by using a log level of Diagnostic or by specifying `vcbuild` or `devenv` in the **Override**

field.

**MSBuild parallel builds:** Specifies the maximum number of projects to build in parallel (optional, up to the maximum number of CPU cores available on the system). If blank, MSBuild will use all available cores to build projects in parallel (enter a value of 1 to disable parallel builds). This option applies only when building Visual Studio 2008 thru 2015 and Delphi Prism solutions with MSBuild.

*Note:* For Visual C++ 8.0 (VS 2005) and later solutions, parallel builds can also be performed by choosing or entering `devenv` in the **Override** field below. The maximum number of projects built in parallel will be determined by the value of *Tools | Options | Project and Solutions | Build and Run | maximum number of parallel project builds* in the Visual Studio IDE.

**Additional command-line options:** This action invokes the `devenv.com`, `msbuild.exe`, or `vcbuild.exe` command-line program to perform builds, and this option can be used to specify additional flags to pass to the tool. Details on the DEVENV flags are covered [here](#). The values entered in this field are passed through directly to the command-line program.

**Override default DEVENV/MSBUILD/VCBUILD location:** If this field is blank, the action will automatically locate the correct `devenv.com` or `msbuild.exe` compiler, based on the version of the project or solution being built. For Visual Studio 2005 thru 2015 and Delphi Prism, this action locates and calls `msbuild.exe` (installed with the .NET Framework 2.0 or later); for Visual Studio 2002/2003, it invokes the appropriate `devenv.com` compiler for the specified project or solution version.

*Notes:*

- For Visual Studio 2005 thru 2015 or Delphi Prism, enter `devenv` in this field to force the project or solution to be built with `devenv.com` rather than `msbuild.exe`. (required for solutions containing Setup & Deployment or Intel C++ Compiler projects, to workaround issues for projects that MSBuild fails to build, or to build VC++ projects in parallel).
- To explicitly specify the compiler executable to use, enter a full path and filename for `devenv.com`, `msbuild.exe`, or `vcbuild.exe` in this field to use that executable rather than the default.
- To build with Xoreax [IncrediBuild](#) (v2.0 and later), enter the full path of `BuildConsole.exe` (usually `C:\Program Files\Xoreax\IncrediBuild\BuildConsole.exe`), and enter any IncrediBuild-specific switches in the *Additional command-line options* field.
- To call the 64-bit version of `msbuild.exe` on 64-bit Windows from the 32-bit edition of Visual Build, enter the full path and filename of `msbuild.exe` in the desired Framework64 folder (i.e., `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe`). To call the 32-bit version of `msbuild.exe` when running the 64-bit edition of Visual Build, enter the full path and filename of `msbuild.exe` in the desired Framework folder (i.e., `C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe`).

### 3.8.4 MSBuild

This action creates a step to perform a Microsoft MSBuild build. When the step is built, the action invokes the MSBuild command-line executable to build the specific project file.

**Project Tab**

**Properties Tab**

**Output Tab**

**Output (More) Tab**

**Options Tab**

## Advanced Tab

### Remote Tab

This action has been tested with MSBuild 2.0 through 12.0 and may work with other versions as well.

*Note:* The Make VS .NET action can also be used to build Visual Studio project and solutions with MSBuild (with additional capabilities).

#### 3.8.4.1 Project Tab

This tab of the MSBuild action configures information about the project to build.

**Project file:** The project file containing the build definitions (optional). If blank, MSBuild will search for a file in the current directory that has an extension containing "PROJ". If one is found, MSBuild will use it as the project file.

**Ignore project file extensions:** Ignores the specified file extensions when determining which project file to build. Use a semicolon or a comma to separate multiple extensions.

**Targets:** Specifies a list of targets to build when building the project (optional, one per line).

#### 3.8.4.2 Properties Tab

This tab of the MSBuild action specifies properties for the project.

**Maximum parallel builds:** Specifies the maximum number of processors that will be used for concurrent builds (optional). A value of \* indicates to use all available processors, or a numeric value specifies the maximum number of concurrent builds. If left blank, builds will be processed sequentially.

**Property names/values:** A list of properties to override (optional). See [here](#) and [here](#) for a list of common properties.

#### 3.8.4.3 Output Tab

This tab of the MSBuild action configures information about the build outputs.

**Include detailed summary:** If checked, shows detailed information at the end of the build log about the configurations that were built and how they were scheduled to nodes.

**Verbosity level:** Specifies the amount of information written to the event log (optional).

**Logger component:** Specifies a logger to log events from MSBuild (optional, one per line). The syntax is:

```
[<logger class>, <logger assembly>[; <logger parameters>]
```

The logger class syntax is:

```
[<partial or full namespace>.<logger class name>
```

The logger assembly syntax is:

```
{<assembly name>[, <strong name>] | <assembly file>}
```

The logger parameters are optional, and are passed to the logger exactly as you typed them.

Examples:

```
XMLLogger,MyLogger,Version=1.0.2,Culture=neutral  
XMLLogger,C:\Loggers\MyLogger.dll;OutputAsHTML
```



**Parameters for console logger:** Parameters to console logger (optional, can be combined by separating with semicolons).

**Disable default console logger:** Disable the default console logger and do not log events to the console (stdout).

#### 3.8.4.4 Output (More) Tab

This tab of the MSBuild action configures additional information about the build outputs.

**File logger:** Log the build output to a single file in the current directory.

**File log number:** If you don't specify a number, the file logger output file is named msbuild.log. If you specify number, the output file is named msbuildn.log, where n is number. Number can be a digit from 1 to 9. You can use the *Additional logger parameters* option to specify the location of the file and other parameters for the file logger.

**Distributed file logger:** Logs the build output of each MSBuild node to its own file. The initial location for these files is the current directory. By default, the files are named "MSBuildNodeid.log". You can use the *Additional logger parameters* option to specify the location of the files and other parameters.

**Distributed loggers:** Log events from MSBuild, attaching a different logger instance to each node (optional, one per line). The following examples show how to use this option:

```
XMLLogger,MyLogger,Version=1.0.2,Culture=neutral  
MyLogger,C:\My.dll*ForwardingLogger,C:\Logger.dll
```

**Additional logger parameters:** Specifies any extra parameters for the file logger and the distributed file logger. The presence of this switch implies that the corresponding /filelogger[number] switch is present. Number can be a digit from 1 to 9.

#### 3.8.4.5 Options Tab

This tab of the MSBuild action configures additional options.

**Toolset version:** Specifies the version of the Toolset to use to build the project (optional).

**Do not auto-include MSBuild.rsp:** Does not include MSBuild.rsp if checked.

**Validate project:** Validate the project against the default schema (or the schema specified in the next field).

**Schema to validate against:** Specifies a schema file to validate the project again (optional, applies only if Validate is checked).

**Additional command-line options:** Use this field to specify additional flags to pass to MSBuild.

**Override MSBuild executable filename:** If not specified, the action locates the MSBuild executable automatically. If the action is unable to locate the command-line tool or multiple versions are installed, browse to or enter the full drive+path+filename here.

*Note:* To call the 64-bit version of msbuild.exe on 64-bit Windows from the 32-bit edition of Visual Build, enter the full path and filename of msbuild.exe in the desired Framework64 folder (i.e., C:\Windows\Microsoft.NET\Framework64\v4.0.30319\msbuild.exe). To call the 32-bit version of msbuild.exe when running the 64-bit edition of Visual Build, enter the full path and

filename of `msbuild.exe` in the desired Framework folder (i.e., `C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe`).

### 3.8.5 SourceSafe

Visual Build provides the SourceSafe action to automate access to Microsoft Visual SourceSafe. The SourceSafe action creates a step for configuring database operations.

When the step is built, the SourceSafe action invokes the VSS `ss.exe` command-line executable to perform the requested operation. Some options do not apply to all operations and will be excluded from the generated command-line.

#### Database Tab

#### Flags Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with SourceSafe 6.0 and 2005 and may work with other versions as well.

#### Notes:

- See the `SourceSafe.bld` sample for projects utilizing this action.
- See the `ContinuousIntegration.bld` sample for a sample of incorporating SourceSafe into continuous integration builds.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.8.5.1 Database Tab

This tab of the SourceSafe action configures information about the database to use and operation to perform.

**Operation:** The VSS operation to perform (required). Most common operations are available in the drop-down list. Other operations may also work but are not explicitly supported (if the operation does not work correctly via the action, a Run Program action can be created which uses the `SSHHELPER` tool/macro directly -- see below). See the [VSS help](#) for details on the available operations.

**Database:** The path to the SourceSafe database to operate on (required). This must be the full path ( `\\server\share[\path]` or `<drive>:\path1\[path2...]` ) to the folder containing the `srcsafe.ini` file for the database. The Browse button can be used to locate the database.

**Username:** The VSS username to login with. If the username of the currently logged in Windows user also exists in the database, the username and password field can be left blank, and that user identity will be used when logging in.

**Password:** Password of the VSS username in the database.

**Projects and files to process:** One or more VSS projects, masks, or filenames to perform the operation on. Project names are prefixed with `$` and use slashes rather than backslashes for folder delimiters.

**Perform operation recursively:** If this option is checked and the previous field contains one or more folders or masks, VSS will search recursively into those folders for matching files.

**Version to operate on:** If left blank, the current (most recent) version of all matching files in the database will be used. To operate on a specific version number, enter the version number; to process all files on or before a given date, enter a string in the format `Dmm/dd/YY`; to process all files as of a given label, enter `LMyLabelName` in this field.

**Path for local files or label to apply:** For all operations except Label, this specifies the drive+path to place files that are retrieved from the database; if the field is left empty, the path used is determined by some SourceSafe Explorer [options](#) (for reliability, especially with automated builds, it is highly recommended that the path or a macro containing the path be specified here). For Label operations, enter the label to apply in this field.

### 3.8.5.2 Flags Tab

This tab of the SourceSafe action configures flags that apply to the command.

**Answer No to all prompts:** The user cannot interactively respond to prompts issued by the VSS command-line tool (and this would interfere with automated builds anyway), so all prompts can automatically be answered with a *No* or *Yes* response, depending on whether this field is checked or not.

**Make local files non-checked out files writeable:** Normally when getting files that are not checked out, the files are marked as read-only as a reminder that the file should be checked out before modifying. To override this behavior and make these files writeable when getting, check this field.

**Handling of writeable local files:** Determines how to treat local files that are different than the version in SourceSafe. *Replace* will overwrite the local file with the version from the database -- be careful with this option since it will also overwrite any writeable files that have been modified (including those that are checked out); the VSS command-line tool doesn't distinguish between checked out and non-checked out writeable files. *Skip* will skip any local writeable files, and not replace with the version from the database (the action will also fail in this case since the VSS command-line tool will return a non-zero exit code). *Merge* will attempt to merge changes between the local and database versions of the file when getting (not recommended).

**Set date/time of local files:** Specifies the timestamp place on files that are retrieved from the database. *Current* will give each file the current local computer's date/time; *Modification* will set the date/time to match the timestamp of the file version that is retrieved (recommended), and *Checkin* will set the date/time to the date/time when the file was checked into SourceSafe.

**File comparison method:** Determines the file comparison method that SourceSafe uses when comparing the database file with the local file to determine if it is different and needs to be retrieved from the database. *Contents* compares the actual contents of the files (can be very slow); *Date/Time* compares only the timestamps of the two files, and *Checksum* compares a (usually) unique file checksum (recommended).

**Command output to display:** Determines how much output to display. The VSS command-line tool can be rather verbose when using the *All* setting (when retrieving files, it lists all matching projects and files, even ones that are up-to-date and don't need to be retrieved [files that are retrieved are prefixed with *Getting* in the output]), and the *Errors only* option may be preferred in such cases.

**Show filenames only, no projects:** When performing a Checkout operation, this option can be checked to prevent each matching project from being logged.

**Show output but do not get files:** This option applies to all operations except Label, and can be useful for debugging purposes. It will display all files that would be retrieved without actually retrieving

the files onto the local computer.

**Treat exit code of 1 as successful:** Some commands return an exit code of 1 to indicate a milder sort of failure:

- Running a Dir command and no items are found.
- Running a Status command and at least one item is checked out.
- Running a Diff and at least one file is different.

Normally, the action will fail in these circumstances; to cause the step to succeed instead, check this option.

### 3.8.5.3 Options Tab

This tab of the SourceSafe action configures additional options.

**Use long filename mode:** If checked, all files are operated on using their long filenames. If unchecked, short filenames are used (not recommended).

**Comment to apply:** Specifies a comment to be applied to the operation and stored in the database for that version/label.

**Additional command-line options:** Use this field to enter any additional ss.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the [VSS help](#) for details on the available command-line flags).

**Specify the ss.exe executable filename:** If not specified, the action locates ss.exe by looking for the default registry value under  
HKEY\_CLASSES\_ROOT\CLSID\{783CD4E4-9D54-11CF-B8EE-00608CC9A71F}\InprocServer32 and combines its path and ss.exe (re-registering ssapi.dll in the Visual Studio/SourceSafe installation path will reinitialize this value). If the action is unable to locate the ss.exe command-line tool or multiple versions are installed, check the checkbox above this field and enter the full drive+path+filename to ss.exe here.

## 3.8.6 Team Build

This action creates a step to perform a Microsoft [Team Foundation Build](#) build. When the step is built, the action invokes the Team Foundation Build command-line executable to perform the requested action.

### Build Tab

### v2008+ Tab

### Options Tab

### Advanced Tab

### Remote Tab

When the step is built, the following temporary macros are created or updated:

- **TEAMBUILD\_BUILDID:** Builder number parsed from Team Build output.

This action has been tested with Visual Studio Team System/Team Foundation Build 2005 thru 2015 and may work with other versions as well (some options require newer versions of Team Build).

*Notes:*

- See the Team System.bld sample for a project utilizing this action.
- The Continuous Integration topic describes the inverse process (invoking Visual Build from Team Foundation Build).

### 3.8.6.1 Build Tab

This tab of the Team Build action configures information about the project and server to build.

**Command:** Specifies the Team Foundation Build command to execute (default is Start).

**Repository server URL:** The repository URL where the solutions being built are checked in (required).

**Team project:** The team project name that has the solutions to be built (required).

**Build type/definition:** The build type or name of build definition in the team project to be used for the build (optional).

**Build machine/Build number:** If the command is "Start," this is the machine where the solutions need to be built. By default the build machine provided in the Build type field will be used. If the command is "Stop" or "Delete," this is the Build number(s) to be operated on.

*Note:* For the Start command, this field only applies to Team Foundation Build 2005 and should be blank for Team Foundation Build 2008 and later.

**Build directory:** The directory where the build process takes place (optional). By default the build directory specified in the Build type is used.

*Note:* When selecting the build directory, make sure there is enough space to build; insufficient space will lead to aborted builds.

### 3.8.6.2 v2008+ Tab

This tab of the Team Build action configures options specific to Team Foundation Build 2008 and later.

**Insert start request into build queue:** If this option is checked, the build start request is inserted into the build queue (applies only to Start command). *Note:* The TEAMBUILD\_BUILDID macro will not be populated if this option is checked.

**Priority:** Specifies the build queue priority (optional).

**Files to get:** Specifies the files to get from version control.

**Version of files to build:** Specifies the custom version of files to get if Custom is specified in previous field: Date/time (D10/20/2005), Changeset version (C1256), Label (Lmylabel), Latest version (T), or Workspace version (Wworkspacename;owner).

**User requesting build:** User who is requesting the build.

**MSBuild arguments:** Use this field to enter arguments to pass to MSBuild (optional).

### 3.8.6.3 Options Tab

This tab of the Team Build action configures additional options.

**Drop location:** Drop location to use for the build (optional).

**Shelveset to include in the build:** Include a shelveset into the build (optional).

**Checkin shelveset after successful build:** If checked, checks in the shelveset after a successful build.

**Additional command-line options:** Use this field to enter any additional Team Foundation Build command-line options to be added to the call that is constructed (optional). See the [Team Build help](#) for details on the available command-line flags.

**Override the Team Build executable filename:** If not specified, the action locates the Team Foundation Build console application automatically. If the action is unable to locate the command-line tool or multiple versions are installed, browse to or enter the full drive+path+filename here.

*Note:* This action uses the **DEVSTUDIO\_NET\_DIR** macro to locate the base directory of the Visual Studio installation path (it appends `Common7\IDE\tfsbuild.exe` to the path in that macro). A global **DEVSTUDIO\_NET\_DIR** macro for the latest version of Visual Studio is initialized during installation and can be updated manually or reset to its default value by clicking the Reload button on the File Locations tab of the Application Options dialog. Or a project or temporary **DEVSTUDIO\_NET\_DIR** macro can be created to specify the path to use.

**Do not log output:** If checked, does not write output while the build is running.

### 3.8.7 Team Foundation

This action creates a step to perform a command on Microsoft [Visual Studio Team Foundation](#) source control server. When the step is built, the action invokes the Team Foundation command-line executable to perform the requested action.

**Command Tab**

**Options Tab**

**Changeset Tab**

**Checkin Tab**

**Difference Tab**

**Dir Tab**

**Get Tab**

**History Tab**

**Label Tab**

**Merge Tab**

**Resolve Tab**

**Shelve Tab**

**Status Tab**

## Workfold Tab

## Workspace Tab

## Advanced Tab

## Remote Tab

This action has been tested with Visual Studio Team Foundation Server 2005 thru 2015 and may work with other versions as well.

*Notes:*

- See the Team System.bld sample for a project utilizing this action.
- See the ContinuousIntegration.bld sample for a sample of incorporating Team Foundation into continuous integration builds.

### 3.8.7.1 Command Tab

This tab of the Team Foundation action configures the source control command to perform and related information.

**Command:** The Team Foundation command to perform (required). Most common operations are available in the drop-down list. See the [Team Foundation Command Line Reference](#) for details on the available commands.

**Old item:** The name and path of the file or folder that is to be renamed or branched (required; applies only to Branch and Rename commands).

**Item spec:** Identifies one or more items, files and folders, destination, shelveset, or workspace to be processed, depending on the command, one per line (required for most commands).

An itemspec, which is short for item specification, is a set of one or more characters that Team Foundation attempts to resolve as an addressable item or set of items, either on your computer or in the source control repository. For all Team Foundation commands that accept an item spec, you can specify either local file system paths such as `c:\projects` or UNC paths such as `\\myshare\projects` or repository paths such as `$/projects/myfiles`. For local paths, you can provide relative paths. For example, if the Local path field is `c:\projects` and you want to check out all items in a subdirectory of the projects folder, you can enter an item spec of `.*` or `abc\*`. If your item spec specifies a repository path, it must be fully-qualified. For example, you cannot check out all items beneath the `$/projects` folder using `./*` as your item spec.

**Recurse:** Whether to recursively process subfolders.

**Local path:** The local drive and path to use when processing the command (required for most commands unless a Set Current Dir action is used to set the path).

**Server:** Identifies the Team Foundation server (optional). This field is required if the command is invoked from a directory that is not mapped to a workspace.

**Username:** Identifies the user to login as (optional) in the format `domain\username` or `username`.

**Password:** Specifies the login user's password (optional).

**Version:** The version of the item for the command to operate on (optional). You can specify a version using the following formats:

- Version number (V2)

- Date/time (D10/20/2005)
- Changeset number (C1256)
- Label (Lmylabel)
- Latest version (T or blank)
- Workspace (Wworkspacename)

**Lock type:** Specifies a lock type or removes a lock from an item:

- None: Removes a lock from an item.
- Checkin: Allows an item to be checked out and edited in all workspaces but prevents users from checking in changes to the item outside this workspace until you explicitly release the checkin lock. If the specified item is locked in any other workspace, the lock operation fails.
- Checkout: Prevents users from checking in or checking out any of the specified items until you explicitly release the lock. If any other users have locked any of the specified items, or if there are existing pending changes against any item, the lock operation fails.

**File type:** Overrides extension-based file type matching and adds files to the repository using the specified type (optional). For more information see the MSDN help topic on [file types](#).

### 3.8.7.2 Options Tab

This tab of the Team Foundation action configures additional command options.

**Workspace:** Specifies the name of the workspace to work in (optional).

**Owner:** Specifies the name of the user who owns the related items (optional; examples: DOMAIN\JuanGo or juango).

**Comment:** A user-provided comment about the command (optional). The comment can be entered directly into the field or the filename of a file containing the comment can be provided.

**Format:** Specifies a verbosity format for any output (brief or detailed).

**No get:** If this option is specified, local copies of the files and folders are not created in your local workspace. *Note:* You can prevent items such as the contents of an /images folder from being retrieved to your workspace during recursive get operations by cloaking a workspace folder.

**No prompt:** Suppresses any dialog boxes or error messages that would otherwise be displayed during the completion of this command.

**Treat partial success as successful:** If checked, causes an exit code of 1 to be treated as success instead of failure.

**Additional command-line options:** Use this field to enter any additional Team Foundation command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the [Team Foundation help](#) for details on the available command-line flags).

**Specify the Team Foundation executable filename:** If not specified, the action locates the Team Foundation console application automatically. If the action is unable to locate the command-line tool or multiple versions are installed, browse to or enter the full drive+path+filename here.

### 3.8.7.3 Changeset Tab

This tab of the Team Foundation action configures additional options for the Changeset command.

**Notes:** Provides one or more notes to associate with the changeset. Enter the notes directly, one on each line in the format



NoteFieldName1=NoteFieldValue1  
NoteFieldName2=NoteFieldValue2

or browse to or enter the filename of a file containing notes in the above format.

**Changeset:** Identifies the changeset to be reviewed or modified (required).

#### 3.8.7.4 Checkin Tab

This tab of the Team Foundation action configures additional options for the Checkin command.

**Author:** Identifies the author of the specified or implied pending changes so that one user can check in changes on behalf of another user (optional). Requires the CheckinOther permission.

**Override reason:** Provides the ability to override a checkin policy failure (optional). This option is only needed when there is a checkin policy and you want to check in anyway. Enter a user-provided description of the reason why the checkin policy is being ignored. If so configured, a notification of the policy override along with this explanation is sent to Team Foundation administrators by email.

**Notes:** Provides one or more notes to associate with the checkin. Enter the notes directly, one on each line in the format

NoteFieldName1=NoteFieldValue1  
NoteFieldName2=NoteFieldValue2

or browse to or enter the filename of a file containing notes in the above format.

#### 3.8.7.5 Difference Tab

This tab of the Team Foundation action configures additional options for the Difference command.

**Shelveset:** Specifies a shelveset to compare to the repository version upon which it was based (optional). This option cannot be combined with an item spec field. To compare individual shelveset items, you can provide a shelveset\_itemspec.

**Options:** Specifies an option string for the difference engine (optional).

**Format:** Specifies an output format.

**Ignore space:** Does not highlight white-space differences between the compared files.

**Ignore case:** Does not highlight differences in letter casing between the compared files.

**Ignore EOL:** Ignores differences between the newline characters in two files or file versions.

#### 3.8.7.6 Dir Tab

This tab of the Team Foundation action configures additional options for the Dir command.

**Folders:** Displays folders only.

**Deleted:** Displays deleted items only.

#### 3.8.7.7 Get Tab

This tab of the Team Foundation action configures additional options for the Get command.

**Preview:** Displays what would happen without actually performing the get command.

**All:** Forces all files to be retrieved, not just those that are out of date.

**Overwrite:** Overwrites writeable files that are not checked out.

**Force:** Implies /all and /overwrite.

### 3.8.7.8 History Tab

This tab of the Team Foundation action configures additional options for the History command.

**Stop after:** Displays history for the number of changesets that you specify.

**Slot mode:** Displays revision history for all items that have ever occupied the specified namespace location.

### 3.8.7.9 Label Tab

This tab of the Team Foundation action configures additional options for the Label command.

**Label name:** Identifies the name of the label to attach, modify, or remove from the specified items.

**Child:** Identifies how to deal with items that have pre-existing labels which are identical to the label you have specified. Use Replace to disregard pre-existing labels. Use Merge to leave the pre-existing labeled items alone.

**Delete:** Removes the label from all specified items.

### 3.8.7.10 Merge Tab

This tab of the Team Foundation action configures additional options for the Merge command.

**Preview:** Shows a preview of the merge.

**Force:** Ignores the merge history and merges the specified changes from the source into the destination, even if some or all of these changes have been merged before.

**Candidate:** Prints a list of all changesets in the source that have not yet been merged into the destination. The list should include the changeset ID that has not been merged and other basic information about that changeset.

**Discard:** Does not actually perform the merge operation, but updates the merge history to reflect that the merge has occurred. You can use this option to instruct Team Foundation to not merge the specified change into the destination.

**Baseless:** Performs a merge in the absence of a basis version.

**No Summary:** Omits summary of errors and warnings when doing so requires more than 10 lines of output.

### 3.8.7.11 Resolve Tab

This tab of the Team Foundation action configures additional options for the Resolve command.

**Auto:** Resolves outstanding conflicts between differing versions of specified items in the current workspace using one of the following options:

- **AcceptMerge:** instructs Team Foundation to reconcile non-overlapping content differences between the specified workspace version of an item and the latest repository version automatically. If Team

Foundation cannot reconcile differences automatically, either because the file is binary or because the workspace and repository versions contain overlapping content changes, the conflict remains unresolved pending the selection of one of the following manual merge options.

- **AcceptTheirs:** instructs Team Foundation to overwrite workspace revisions with the repository revision.
- **AcceptYours:** instructs Team Foundation to keep your changes and discard the changes in the repository version of an item.

**Override/convert:** Specifies either

- an optional encoding that overrides the encoding of the files involved in a three-way merge (Team Foundation saves the resulting merge output in the specified encoding in your workspace) or
- the optional encoding to which each of the inputs in a three-way merge operation should be temporarily converted (Team Foundation saves the resulting merge output in the specified encoding in your workspace).

**Preview:** Displays merge conflicts but does nothing with them.

#### 3.8.7.12 Shelf Tab

This tab of the Team Foundation action configures additional options for the Shelf command.

**Replace:** Replaces the existing shelveset with the same name and /owner as the one you specify.

**Move:** Removes all pending changes from the current or specified workspace upon successful completion of the shelf operation.

**Delete:** Deletes the specified shelveset. Only the repository option may be combined with this option. If you do not check the no prompt option, a confirmation message appears when specified.

#### 3.8.7.13 Status Tab

This tab of the Team Foundation action configures additional options for the Status command.

**Shelveset:** The name of the shelveset for which you would like to see a list of changes (optional). This option cannot be combined with the Workspace field.

**User:** Filters the list of changes to the named user (optional). An asterisk (\*) can be used to denote all users. The default is the current user.

#### 3.8.7.14 Workfold Tab

This tab of the Team Foundation action configures additional options for the Workfold command.

**Operation:** Specifies the type of operation to perform:

- **Map:** Specifies that Team Foundation should associate the given local path with the given repository folder (the default).
- **Unmap:** Specifies that the given folder mapping should be deleted.
- **Cloak:** Specifies that the given folder should be excluded from the workspace view.
- **Decloak:** Uncloaks a folder so that it can be retrieved into the workspace.

#### 3.8.7.15 Workspace Tab

This tab of the Team Foundation action configures additional options for the Workspace command.

**New name:** Renames an existing workspace to the specified value (optional).

**Computer:** Specifies the name of the computer on which to create the workspace (optional).

**New:** Creates a new workspace.

**Delete:** Deletes the specified workspace.

### 3.8.8 Team Test

This action creates a step to run unit tests with Microsoft Visual Studio MSTest or Team Test. When the step is built, the action invokes the MSTest / Team Test command-line executable to perform the requested action.

#### Test Tab

#### Filter Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with Visual Studio 2005 thru 2015 and Visual Studio Team System 2005 thru 2015 and may work with other versions as well (some options require newer versions of Team System).

*Note:* See the Team System.bld sample for a project utilizing this action.

#### 3.8.8.1 Test Tab

This tab of the Team Test action configures information about the test(s) to perform.

**Metadata file:** Browse to or enter a test metadata file that contains the tests to run (optional):

- A test metadata `.vsmdi` file is created for the solution when creating test lists using the Test Manager window. This file contains information about all the tests listed in the Test Manager window. These are all the tests that exist in all test projects in the solution. The test metadata file is an XML file that is created in the solution folder. When specifying a test metadata file, it is recommended to also enter specific tests to run in the Test lists fields.

**Test containers (one per line):** Browse to or enter test container files (optional):

- For ordered tests, the test container is the `.orderedtest` file that defines the ordered test.
- For unit tests, it is the assembly built from the test project that contains the unit test source files.

**Test lists:** Specifies tests list to be run as specified in the metadata file (optional, one per line).

**Run configuration file:** Enter or browse to a run configuration file (optional). A run configuration file can be specified in other ways, such as by using a test metadata file. Specifying a run configuration file here overrides the value in a test metadata file. If not specified here or in the metadata file, the test run uses the default run configuration file.

**Unique:** Checking this field instructs Team Test to run only a single test whose name matches the values in the Test cases field.

### 3.8.8.2 Filter Tab

This tab of the Team Test action configures filtering of the action results.

**Tests to run:** Specify the names of tests to run (optional).

**Detail property names:** Specify the names of properties to show values for in addition to the test outcome (optional). If a property exists for a given test case, its information is included in the output result summary.

**Categories of tests to execute:** Filter tests to run based on the test category (optional). You can use logical operators & (and), | (or), and ! (not) to filter tests.

**Maximum priority:** Only tests whose priority is less than or equal to this value will be executed (optional).

**Minimum priority:** Only tests whose priority is greater than or equal to this value will be executed (optional).

### 3.8.8.3 Team Explorer

This tab of the Team Test action configures Team Explorer related options that apply to the action.

**Publish server:** Publish results to the operational store of the specified server (optional). The correct format for the server depends on whether the Team Foundation Server has been registered on the client computer:

- If the Team Foundation Server is not registered on the client computer, use the URI that identifies the Team Foundation Server. For example, `http://OurTFSMachine:8080`.
- If the Team Foundation Server computer is registered, use a shortened form, namely: `OurTFSMachine`.

**Publish build ID:** Publish test results using the build ID (optional). To publish test data, specify the number of a build that has completed. A build number has the following format, which reflects the time it occurred: `MMDDYY_HHMMSS_nnnnn`. To know which builds are available, perform the following steps:

1. Open Visual Studio and connect to a Team Foundation Server.
2. Open the team project.
3. Expand the nodes for the team project, for Public Builds, and for the build configuration name to display the names of builds that have completed. To obtain the build number, perform the following steps:
4. In the Test Results window, click Publish. *Note:* The Publish button is enabled only when test run results are displayed in the Test Results window.
5. In the Publish dialog box, select a build from the drop-down list and then click OK. Note the name of the build you selected. If the test results publish successfully, use the name of this build as the build ID.

**Publish results file:** Enter or browse to the results filename to publish tests that were run previously (optional). To publish the results of the current run using the default name, leave this field blank.

**Team Project:** The name of the Team Project to which the build belongs.

**Platform:** The platform of the build against which to publish test results (an example is "x86").

**Flavor:** The flavor of the build against which to publish test results (an example is "Release").

**Test configuration management ID:** Test configuration management ID to associate with the published run (optional). Use only with *Publish server* option.

**Test configuration management name:** Test configuration management name to associate with the published run (optional). Use only with *Publish server* option.

#### 3.8.8.4 Options Tab

This tab of the Team Test action configures additional options.

**Test settings file:** Enter or browse to a settings file to use (optional).

**Results file:** Enter or browse to a results file to save the test run results to (optional).

**Additional command-line options:** Use this field to enter any additional Team Test command-line options to be added to the call that is constructed. See the [MSTest help](#) for details on the available command-line flags.

**Override the Team Test executable filename:** If not specified, the action locates the MSTest or Team Test console application automatically. If the action is unable to locate the command-line tool or multiple versions are installed, browse to or enter the full drive+path+filename here.

*Note:* This action uses the **DEVSTUDIO\_NET\_DIR** macro to locate the base directory of the Visual Studio installation path (it appends `Common7\IDE\mstest.exe` to the path in that macro). A global **DEVSTUDIO\_NET\_DIR** macro for the latest version of Visual Studio is initialized during installation and can be updated manually or reset to its default value by clicking the Reload button on the File Locations tab of the Application Options dialog. Or a project or temporary **DEVSTUDIO\_NET\_DIR** macro can be created to specify the path to use.

**No startup message:** If checked, no startup banner and copyright message will be output.

**Run tests within MSTest process:** Improves test run speed but increases risk to the MSTest process.

**Do not save test results:** Do not save the test results in a TRX file. Improves test run speed but does not save the test run results.

#### 3.8.9 VS6 Get Version

This custom action creates a step to retrieve the file version of a Visual Studio 6.0 project or resource file (C++ projects and resource files and Visual Basic projects only). Select the filename in the Project tab, and when the step is built, the file version is retrieved from the project or RC file and stored in the temporary macro **VS6\_PROJ\_VER**.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.8.10 VS.NET Get Version

This custom action creates a step to retrieve the file and assembly versions of a Visual Studio .NET 2002 thru 2015 or Delphi Prism project or individual assembly or resource file (C#, VB.NET, J#, setup, Oxygene, and C++ projects; valid extensions are `.csproj`, `.csdproj`, `.vbproj`, `.vbdproj`, `.vcproj`, `.vcxproj`, `.icproj`, `.vdproj`, `.oxygene`, `.rc*`, `.cs`, `.vb`, `.jsl`, `.cpp`, `.pas`).

When the step is built, the following temporary macros are created or updated:

- **VSNET\_PROJ\_VER**: File version (Product version is returned for setup and deployment projects).
- **VSNET\_PROD\_VER**: Product version.
- **VSNET\_ASSEM\_VER**: Assembly version (for managed projects only).

*Notes:*

- This action has been tested with Visual Studio .NET 2002 through 2015 and Delphi Prism.
- This action does not require Visual Studio .NET to be installed, but it does require the Microsoft [.NET Framework](#) (version 1.1 or later).
- VS.NET defaults the Assembly version to 1.0.\* and that value will be returned by this action rather than the actual version of the compiled assembly. Use the Make VS.NET action's set and increment capabilities to increment and get the true assembly version.

### 3.8.11 Visual Studio Integration

Visual Build integrates with Microsoft Visual Studio via add-ins:

#### Visual Studio 2002 - 2013 Add-In

An add-in for launching Visual Build is added to the Visual Studio .NET 2002 thru 2013 IDE during installation. To enable the add-in within Visual Studio, choose *Tools | Add-In Manager* on the menu and check *Visual Build VS <ver> Add-in* (also check the Startup checkbox to make the add-in available each time the IDE is started). Then, Visual Build can be launched by choosing *Tools | Visual Build* from the Visual Studio menu bar. A new Make VS.NET action will be created and pre-populated with the filename for the current workspace. If the *Reload last project* option is checked, the new step will be inserted into the last-opened .bld file. If this option is unchecked or the Shift key is held down when clicking on Visual Build in Visual Studio, the new step will be inserted into a new project. *Note:* For Visual Studio 2008+, if the .bld file is saved in the same path as the .sln file with the same base name (i.e., c:\full\path\xyz.bld and c:\full\path\xyz.sln), the add-in will open that .bld file instead of opening the previous .bld file (or creating a new project) and inserting a new step.

#### Visual Studio 2015 Extension

An extension for launching Visual Build can be added to the Visual Studio .NET 2015 IDE during installation. To enable the extension within Visual Studio, choose *Tools | Extensions and Updates* on the menu, find Visual Build and click Enable, then restart Visual Studio. Visual Build can be launched by choosing *Tools | Visual Build* from the Visual Studio menu bar. A new Make VS 2015 action will be created and pre-populated with the filename for the current workspace. If the *Reload last project* option is checked, the new step will be inserted into the last-opened .bld file. If this option is unchecked or the Shift key is held down when clicking on Visual Build in Visual Studio, the new step will be inserted into a new project. *Note:* If the .bld file is saved in the same path as the .sln file with the same base name (i.e., c:\full\path\xyz.bld and c:\full\path\xyz.sln), the add-in will open that .bld file instead of opening the previous .bld file (or creating a new project) and inserting a new step.

#### Visual Basic 6.0 Add-In

After Visual Build is installed, a toolbar button for Visual Build is added to the Visual Basic 6.0 IDE. Visual Build can be started by clicking the button or by choosing *Add-Ins | Visual Build* from the menu bar. A new Make VB6 action will be created and pre-populated with the filename for the current project or group.

#### Visual C++ 6.0 Add-In

A toolbar button for Visual Build is added to the Visual C++ 6.0 IDE during installation. Visual Build can be started by clicking the button (a keystroke can also be associated with the wizard by selecting *Tools | Customize | Keyboard* from the menu, choosing a Category of Add-ins, selecting *VisBuildProCommand* from the list, entering a keystroke in the *Press new shortcut key* field, and clicking Assign). A new Make VC6 action will be created and pre-populated with the filename for the current workspace.

### 3.8.12 Visual Studio & .NET Macros

The following global macros are defined for Microsoft Visual Studio and .NET Framework integration:

**DOTNET\_DIR**

The path where the Microsoft .NET Framework (highest version) is installed.

**DOTNETSDK\_DIR**

The path where the Microsoft .NET Framework SDK (highest version) is installed. Most [SDK tools](#) can be called by using a Run Program step with a Command like %DOTNETSDK\_DIR%\bin\TlbImp.exe or %DOTNETSDK\_DIR%\bin\x64\TlbImp.exe for the 64-bit binaries. Custom actions for many SDK tools are also available.

**DEVSTUDIO\_NET\_DIR**

The base path where Visual Studio (.NET 2002 thru 2015; highest version) is installed.

**NMAKE\_NET**

Defines the NMAKE command-line make tool included with the Microsoft .NET Framework SDK, which can be used to build .NET Framework projects. Enter a step command of %NMAKE\_NET% /? to see the available syntax, and view [this link](#) for more details.

## 3.9 Microsoft .NET

### 3.9.1 App Packager

The App Packager action calls MakeAppx to pack or unpack a Windows 8 application store application package.

**Package Tab****Options Tab****Advanced Tab****Remote Tab**

#### 3.9.1.1 Package Tab

This tab of the App Packager action specifies package settings.

**Action:** The action to perform (required).

**Filename:** The filename of the package (required).

**Content/output directory:** The content directory if packing or the output directory if unpacking (required unless specifying a mapping file when packing).

**Mapping file:** The mapping file to use to create the package (optional). The first line contains the string [Files], and the lines that follow specify the source (disk) and destination (package) paths in quoted strings.

**Hash algorithm:** The hash algorithm to use for creating the block map (optional, default is SHA256).

**Overwrite existing files:** If checked, forces the output to overwrite existing files.

**Do not overwrite existing files:** If checked, prevents the output from overwriting existing files



(mutually exclusive with overwrite option).

**Verbose mode:** Logs verbose message output.

### 3.9.1.2 Options Tab

This tab of the App Packager action configures additional options.

**Disable validation of files declared in manifest:** Disables validation that checks whether files exist in the package as specified in the manifest. Use this option when files declared in the manifest have variations or resource versions for scale, language, etc.

**Skip validation ensuring package is installable:** Skips semantic validation that ensures the package will be installable on Windows.

**Unpack to folder under output directory named after package:** In the .NET Framework version 2.0, displays additional information in MSIL verification messages.

**Override executable filename:** Overrides the default executable (MakeAppx.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of MakeAppx.exe, and the 64-bit edition of Visual Build calls the 64-bit version of MakeAppx.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

## 3.9.2 Export Type Library

The Export Type Library action creates a step to call TlbExp to generate a COM type library that describes types defined in a .NET assembly.

**Assembly filename:** The assembly for which to export a type library (required).

**Type library filename:** Specifies the name of the type library file to generate (optional). If blank, the action generates a type library with the same name as the assembly (the actual assembly name, which might not necessarily be the same as the file containing the assembly) and a .tlb extension.

**Capitalization of names in type library:** Specifies the capitalization of names in a type library (optional). Each line specifies the capitalization of one name in the type library.

**Override executable filename:** Overrides the default executable (TlbExp.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of TlbExp.exe, and the 64-bit edition of Visual Build calls the 64-bit version of TlbExp.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

## 3.9.3 GAC Install

This action creates a step to install or uninstall a .NET assembly from the Global Assembly Cache (GAC). When installing, the Assembly field should be the filename of the assembly; when unregistering, it must be the name of the assembly, optionally including the version, culture, and public key token (i.e., MyAssembly or MyAssembly,Version=1.1.0.0,Culture=en,PublicKeyToken=874e23ab874e23ab).

**Assembly:** The filename or assembly name of the assembly to install or uninstall. When uninstalling an assembly, if Force is unchecked, the assembly name can be a partial name for matching multiple assemblies. This action can also be used in conjunction with the Process Files action to process multiple assemblies.

**Uninstall the assembly:** Determines whether the assembly is installed or uninstalled from the GAC.

**Uninstall from NGEN cache:** If uninstalling, determines whether the assembly is also uninstalled from the NGEN cache.

**Force overwrite/uninstall:** Determines if an existing file will be overwritten if installing and the file already exists, or if uninstalling, whether all install references will be removed.

**Update AssemblyFolders registry key:** If a folder name is specified in this field, a registry key by that name is create/updated or removed from `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ .NETFramework\AssemblyFolders` and its default entry set to the path of the assembly file. This allows the assembly to be found when compiling other .NET components that reference the assembly, and causes the assembly to automatically show up in the list of references in Visual Studio .NET without the user having to browse to it.

**Override executable filename:** Overrides the default executable (gacutil.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of GacUtil.exe, and the 64-bit edition of Visual Build calls the 64-bit version of GacUtil.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the gacutil.exe call that is constructed (optional).

When built, this action invokes the [GACUtil](#) command-line tool to install or uninstall the assembly (the [.NET Framework SDK](#) needs to be installed wherever this action is built), and also updates the registry key above if a folder is provided for the AssemblyFolders field.

### 3.9.4 Generate Resource Files

The Generate Resource Files action creates a step that calls ResGen to convert .txt and .resx files to CLR .resources, .resx, or text files.

#### Files Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

##### 3.9.4.1 Files Tab

This tab of the Generate Resource Files Action action specifies input and output files.

**Input + output files:** The names of one or more input and output files (required).

For input files, the extension must be one of the following:

- .txt: Specifies the extension for a text file to convert to a .resources or a .resx file. Text files can only contain string resources.
- .resx: Specifies the extension for an XML-based resource file to convert to a .resources or a .txt file.
- .resources: Specifies the extension for a resource file to convert to a .resx or a .txt file.

Output filenames are optional when converting from a .txt or .resx file. You can specify the .resources extension when converting a text or .resx file to a .resources file. If you do not specify an output file, the action appends a .resources extension to the input filename argument and writes the file to the directory that contains filename.

The output filenames are required when converting from a .resources file. Specify the .resx extension when converting a .resources file to an XML-based resource file. Specify the .txt extension when converting a .resources file to a text file. You should only convert a .resources file to a .txt file when the .resources file contains only string values.

#### 3.9.4.2 Options Tab

This tab of the Generate Resource Files Action action configures additional options.

**Use source file directory as current directory for resolving relative paths:** Self-explanatory.

**Load types from assemblies:** Specifies assembly filenames (one per line) to load referenced types from.

**Override executable filename:** Overrides the default executable (ResGen.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of ResGen.exe, and the 64-bit edition of Visual Build calls the 64-bit version of ResGen.exe if found.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

### 3.9.5 Import Type Library

The Import Type Library action calls Tblmp to convert type definitions in a COM type library into equivalent definitions in a .NET assembly.

#### Import Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.9.5.1 Import Tab

This tab of the Import Type Library action specifies the general settings.

**Type library filename:** The name of any file that contains a COM type library (required).

**Output filename:** Specifies the name of the output file, assembly, and namespace in which to write the metadata definitions (optional). This option has no effect on the assembly's namespace if the type library specifies the Interface Definition Language (IDL) custom attribute that explicitly controls the assembly's namespace. If you do not specify this option, the action writes the metadata to a file with the same name as the actual type library defined within the input file and assigns it a .dll extension. If the output file is the same name as the input file, the tool generates an error to prevent overwriting the type library.

**Assembly namespace:** Specifies the namespace in which to produce the assembly (optional).

**Version:** Specifies the version number of the assembly to produce (optional). Specify in the format *major.minor.build.revision*.

**Produce primary interop assembly:** Produces a primary interop assembly for the specified type library. Information is added to the assembly indicating that the publisher of the type library produced the assembly. By specifying a primary interop assembly, you differentiate a publisher's assembly from any other assemblies that are created from the type library using Tlbimp.exe. You should only use this option if you are the publisher of the type library that you are importing. Note that you must sign a primary interop assembly with a strong name.

**Produce interfaces without .NET security checks:** Produces interfaces without .NET Framework security checks. Calling a method that is exposed in this way might pose a security risk. You should not use this option unless you are aware of the risks of exposing such code

**Import a COM style SafeArray as a managed Array class:** Specifies to the tool to import a COM style SafeArray as a managed System.Array Class type.

### 3.9.5.2 Options Tab

This tab of the Import Type Library action configures additional options.

**Sign with key pair found in key container:** Signs the resulting assembly with a strong name using the public/private key pair found in the specified key container (optional).

**Sign with key pair found in file:** Signs the resulting assembly with a strong name using the publisher's official public/private key pair found in filename (optional).

**Sign with public key in file:** Specifies the file containing the public key to use to sign the resulting assembly (optional). If you specify the key container or key file option instead, the action generates the public key from the public/private key pair supplied there. This option supports test key and delay signing scenarios.

**Delay sign:** Specifies to sign the resulting assembly with a strong name using delayed signing. You must specify this option with either the key container, key file, or public key option. For more information on the delayed signing process, see Delay Signing an Assembly.

**Do not display copyright:** Suppresses display of version and copyright.

**Override executable filename:** Overrides the default executable (Tlbimp.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of Tlbimp.exe, and the 64-bit edition of Visual Build calls the 64-bit version of Tlbimp.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

### 3.9.6 Install .NET Services

The Install .NET Services action calls RegSvcs to register, install, and configure .NET COM+ services.

#### Services Tab

#### Options tab

#### Advanced Tab

## Remote Tab

### 3.9.6.1 Services Tab

This tab of the Install .NET Services action specifies general settings.

**Assembly filename:** The source assembly file (required). The assembly must be signed with a strong name.

**Behavior:** The behavior or action to perform.

**COM+ application name:** Specifies the name of the COM+ application to either find or create (optional).

**Expect an existing application:** Self-explanatory.

**Do not reconfigure an existing application:** Self-explanatory.

**Configure components only:** Configures components only; ignores methods and interfaces.

**Type library file to install:** Specifies the type library file to install (optional).

**Use an existing type library:** Self-explanatory.

### 3.9.6.2 Options Tab

This tab of the Install .NET Services action configures the SQL statement to be executed and how to process the results.

**Do not display copyright:** Suppresses display of version and copyright.

**Quiet mode:** Suppresses logo and success message display.

**Override executable filename:** Overrides the path+filename of the regsvcs.exe to call (optional). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of regsvcs.exe, and the 64-bit edition of Visual Build calls the 64-bit version of regsvcs.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

## 3.9.7 Manifest Generator

The Manifest Generator action calls mage.exe to create or edit application and deployment manifests.

### Package Tab

### Settings Tab

### Signing Tab

### Options Tab

### Advanced Tab

### Remote Tab

### 3.9.7.1 Manifest Tab

This tab of the Manifest Generator action specifies package settings.

**Command:** The command to perform (required).

**Filename:** The manifest filename (required). For the New command, specifies the type (deployment or application).

**Application name:** Identity of the application (optional). ClickOnce will use this name to identify the application in the Start menu (if the application is configured to install itself) and in Permission Elevation dialog boxes. If no name is supplied, the default is "deploy".

**Version:** The version of the deployment (optional, deployment and application manifests only). Must be a valid version string of the format "N.N.N.N", where "N" is an unsigned 32-bit integer. The default value is "1.0.0.0".

**Processor:** The microprocessor architecture on which this distribution will run (optional, deployment and application manifests only). Required if you are preparing one or more installations whose assemblies have been precompiled for a specific microprocessor. Default is msil, or Microsoft Intermediate Language, which means all of your assemblies are platform-independent, and the common language runtime (CLR) will just-in-time compile them when your application is first run.

**Application manifest file:** Inserts a reference to a deployment's application manifest into its deployment manifest (optional, deployment manifests only).

### 3.9.7.2 Settings Tab

This tab of the Manifest Generator action specifies general settings.

**Minimum version allowed:** The minimum version of this application a user can run (optional, deployment manifests only). This option makes the named version of your application a required update; if you release a version of your product with an update to a breaking change or a critical security flaw, you can use this flag to specify that this update must be installed, and that the user cannot continue to run previous versions. If no version is specified, Mage.exe will use the version listed in the ClickOnce deployment manifest as specified by the -Version flag.

**Reference URI:** Inserts a URL or file path reference to the application manifest file (optional, deployment manifests only). This file must be the full path to the application manifest.

**Publisher name:** The name of the publisher (optional).

**Support URL:** Application support URL (optional).

**Application update URL:** Specifies the URL which ClickOnce will examine for application updates (optional, deployment manifests only).

**Include provider URL:** Whether to include the application update URL.

**Install application on local machine:** Indicates whether or not the ClickOnce application should install onto the local machine, or whether it should run from the Web (deployment manifests only). Installing an application gives that application a presence in the Windows Start menu. If you specify the Minimum version option and a user has a version less than that installed, it will force the application to install, regardless of whether this is checked.

**WPF browser app:** Indicates whether this is a WPF browser application (deployment manifests only).

### 3.9.7.3 Signing Tab

This tab of the Manifest Generator action specifies signing options.

**Certificate file:** Specifies The location of a digital certificate for signing a manifest (optional).

**Password:** The password used for signing a manifest with a digital certificate. Required if certificate file is provided.

**Certificate hash:** The hash of a digital certificate stored in the personal certificate store of the client computer. This corresponds to the Thumbprint property of a digital certificate viewed in the Windows Certificates Console. The value can be either uppercase or lowercase, and can be supplied either as a single string or with each octet of the Thumbprint separated by spaces and the entire Thumbprint enclosed in quotes.

**Hash algorithm:** The hash algorithm (optional).

**Timestamp URI:** The URI of the timestamp server to use to timestamp the signature.

### 3.9.7.4 Options Tab

This tab of the Manifest Generator action configures additional options.

**Populate from directory:** Populates the application manifest with descriptions of all assemblies and files found in the path to directory containing the application you want to deploy (application manifests only). For each file in the directory, Mage.exe decides whether the file is an assembly or a static file. If it is an assembly, it adds a <dependency> tag and installFrom attribute to the application with the assembly's name, code base, and version. If it is a static file, it adds a <file> tag. Mage.exe will also use a simple set of heuristics to detect the main executable for the application, and will mark it as the ClickOnce application's entry point in the manifest. Mage.exe will never automatically mark a file as a "data" file; this must be done manually.

Mage.exe also generates a hash for each file based on its size. ClickOnce uses these hashes to ensure that no one has tampered with the deployment's files since the manifest was created. If any of the files in your deployment change, you can run Mage.exe with the Update command and this option, and it will update the hashes and assembly versions of all referenced files.

This option will include all files in all subdirectories found within the path.

If you use this option with the Update command, Mage.exe will remove any files in the application manifest that no longer exist in the directory.

**Output filename:** Specifies the output path of the file that has been created or modified (optional). If not supplied when using the New command, the output is written to the current working directory with a default file name, whose exact name depends upon the type of file you create:

- Deployment: deploy.application
- Application: application.exe.manifest

If this option is not supplied when using the Update command, Mage.exe will write the file back to the input file.

**Trust level:** Specifies the trust level of the application.

**Use manifest for trust:** Whether to use a manifest for the trust info.

**Override executable filename:** Overrides the default executable (mage.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of mage.exe, and the 64-bit edition of Visual Build

calls the 64-bit version of mage.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

### 3.9.8 NuGet

The NuGet action calls NuGet to add, remove, and update libraries and tools in Visual Studio projects that use the .NET Framework.

This action has been tested with versions 1 thru 3 of NuGet and may also work with other versions.

#### Command Tab

#### Settings Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

##### 3.9.8.1 Command Tab

This tab of the NuGet action defines the command to perform.

**Command:** The command to perform (required).

**Package ID / path / project:** Specifies the package ID, path, or project (optional).

**Output directory:** Specifies the directory in which packages will be installed.

**Package sources:** A list of packages sources to use for the install (one per line).

**Version:** The version of the package to install.

**Do not prompt:** Do not prompt for user input or confirmations.

##### 3.9.8.2 Settings Tab

This tab of the NuGet action defines additional settings.

**Output verbosity:** Display the specified amount of details in the output.

**MSBuild version:** Specifies the version of MSBuild to be used. By default the MSBuild in your path is picked, otherwise it defaults to the highest installed version of MSBuild.

**File conflict action:** The action to take when asked to overwrite or ignore existing files referenced by the project.

**Minor updates only:** Looks for updates with the highest version available within the same major and minor version as the installed package.

**Allow updating to pre-release versions:** Allows updating to pre-release versions. Not required when updating prerelease packages that are already installed (applies only to update command).

**Update running NuGet.exe:** Update the running NuGet.exe to the newest version available from the



server (applies only to update command).

**API key:** The API key for the server.

**Configuration file:** The NuGet configuration file. If not specified, file %AppData%\NuGet\NuGet.config is used as configuration file.

### 3.9.8.3 Options Tab

This tab of the NuGet action defines additional options.

**Override executable filename:** Overrides the default NuGet executable. If not specified, the NuGet.exe command-line executable must be found in the PATH environment variable.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

## 3.9.9 PEVerify

The PEVerify action calls PEVerify to determine whether .NET MSIL code and associated metadata meets type safe requirements.

### Verify Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

### 3.9.9.1 Verify Tab

This tab of the PEVerify action specifies general settings.

**Portable executable filename:** The portable executable (PE) file for which to check the MSIL and metadata (required).

**Perform MSIL type safety verification checks for methods:** Performs MSIL type safety verification checks for methods implemented in the assembly specified by filename.

**Perform metadata validation checks:** Performs metadata validation checks on the assembly specified by filename. This walks the full metadata structure within the file and reports all validation problems encountered.

**Display error codes in hex format:** Displays error codes in hexadecimal format.

**Ignore repeating error codes:** Self-explanatory.

**Ignore the specified error codes:** Ignores the specified error codes (optional, separate with commas).

**Abort verification after max errors:** Aborts verification after the specified number of errors.

### 3.9.9.2 Options Tab

This tab of the PEVerify action configures additional options.

**Report verification times in milliseconds:** Measures and reports the following verification times in

milliseconds.

**Quiet mode:** Specifies quiet mode; suppresses output of the verification problem reports. The action still reports whether the file is type safe, but does not report information on problems preventing type safety verification.

**Verbose mode:** In the .NET Framework version 2.0, displays additional information in MSIL verification messages.

**Override executable filename:** Overrides the default executable (PEVerify.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of PEVerify.exe, and the 64-bit edition of Visual Build calls the 64-bit version of PEVerify.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

### 3.9.10 Strong Name Tool

The Strong Name Tool action creates a step that calls SN.exe to perform commands on signed .NET assemblies.

**Command to perform:** The SN command to execute.

**Key container/CSP:** Specifies the key container or Cryptographic Service Provider for the command (required for some commands).

**Input filename/assembly:** The input or first filename or assembly for the command (required for some commands).

**Output filename/assembly:** The output or second filename or assembly for the command (required for some commands).

**Display public key in addition to token:** Displays the public key as well as the token (applies only to display commands).

**Recompute hashes for all files in assembly:** Recomputes hashes for all files in the assembly (applies only to re-sign commands).

**Override executable filename:** Overrides the default executable (SN.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of SN.exe, and the 64-bit edition of Visual Build calls the 64-bit version of SN.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

### 3.9.11 WSDL

The WSDL action calls WSDL to generate .NET code for XML web services and web service clients.

#### WSDL Tab

#### Authentication Tab

#### Options Tab

#### Advanced Tab

## Remote Tab

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.9.11.1 WSDL Tab

This tab of the WSDL Action action specifies general settings.

**WSDL URL or filename:** The URL to a WSDL contract file (.wsdl), XSD schema file (.xsd), or discovery document (.disco). Note that you cannot specify a URL to a .discomap discovery document, or the path to a local WSDL contract file (.wsdl), XSD schema file (.xsd), or discovery document (.disco or .discomap). Required.

**File to save generate proxy code to:** Specifies the file in which to save the generated proxy code (optional). The tool derives the default file name from the XML Web service name. The tool saves generated datasets in different files.

**Base URL to use when calculating URL fragment:** Specifies the base URL to use when calculating the URL fragment (optional). The tool calculates the URL fragment by converting the relative URL from this field to the URL in the WSDL document. You must specify the previous field with this option.

**Configuration key to read default value for URL when generating code:** Specifies the configuration key to use in order to read the default value for the URL property when generating code (optional).

**Language to use for generated proxy class:** Specifies the language to use for the generated proxy class (optional). You can specify CS (C#, default), VB (Visual Basic), JS (Jscript) or VJS (Visual J#) as the language argument. You can also specify the fully qualified name of a class that implements the System.CodeDom.Compiler.CodeDomProvider Class.

**Protocol to implement:** Specifies the protocol to implement (optional). You can specify SOAP (default), HttpGet, HttpPost, or a custom protocol specified in the configuration file.

### 3.9.11.2 Authentication Tab

This tab of the WSDL Action specifies authentication options.

**Domain name:** Specifies the domain name to use when connecting to a server that requires authentication (optional).

**User name:** Specifies the user name to use when connecting to a server that requires authentication (optional).

**Password:** Specifies the password to use when connecting to a server that requires authentication (optional).

**Proxy server:** Specifies the URL of the proxy server to use for HTTP requests. The default is to use the system proxy setting (optional).

**Proxy domain:** Specifies the domain to use when connecting to a proxy server that requires authentication (optional).

**Proxy user name:** Specifies the user name to use when connecting to a proxy server that requires authentication (optional).

**Proxy password:** Specifies the password to use when connecting to a proxy server that requires

authentication (optional).

### 3.9.11.3 Options Tab

This tab of the WSDL Action configures additional options.

**Namespace for generated proxy:** Specifies the namespace for the generated proxy or template. The default namespace is the global namespace (optional).

**Display errors in error reporting format:** Displays errors in a format similar to the error reporting format used by language compilers.

**Generate abstract class:** Generates an abstract class for an XML Web service based on the contracts. The default is to generate client proxy classes.

**Quiet mode:** Suppresses the startup banner display.

**Override executable filename:** Overrides the default executable (WSDL.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of WSDL.exe, and the 64-bit edition of Visual Build calls the 64-bit version of WSDL.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

## 3.9.12 XSD

The XSD action calls XSD to generate XML schema or CLR classes from XDR, XML, and XSD files or from classes in a .NET assembly.

### XSD Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.9.12.1 XSD Tab

This tab of the XSD Action specifies general settings.

**Input filename:** The input file to convert (required). You must specify the extension as one of the following: .xdr, .xml, .xsd, .dll, or .exe.

If you specify an XDR schema file (.xdr extension), the action converts the XDR schema to an XSD schema. The output file has the same name as the XDR schema, but with the .xsd extension.

If you specify an XML file (.xml extension), the action infers a schema from the data in the file and produces an XSD schema. The output file has the same name as the XML file, but with the .xsd extension.

If you specify an XML schema file (.xsd extension), the action generates source code for runtime objects that correspond to the XML schema.

If you specify a runtime assembly file (.exe or .dll extension), the action generates schemas for one or more types in that assembly.

**Directory for output files:** Specifies the directory for output files (optional). This argument can appear only once. The default is the current directory.

**XSD output type:** Classes: Generates classes that correspond to the specified schema. To read XML data into the object, use the System.XML.Serialization.XMLSerializer.Deserialize method. DataSet: Generates a class derived from DataSet that corresponds to the specified schema. To read XML data into the derived class, use the System.Data.DataSet.ReadXml method.

**Language to use for generate proxy class:** Specifies the programming language to use (optional). Choose from CS (C#; default), VB (Visual Basic), JS (Jscript), or VJS (Visual J#). You can also specify a fully qualified name for a class implementing System.CodeDom.Compiler.CodeDomProvider

**Namespace for generated proxy or template:** Specifies the runtime namespace for the generated types. The default namespace is Schemas.

**URI for elements in the schema to generate code for:** Specifies the URI for the elements in the schema to generate code for (optional). This URI, if present, applies to all elements specified with the Elements option.

### 3.9.12.2 Options Tab

This tab of the XSD Action configures additional options.

**Element in schema to generate code for:** Specifies elements in the schema to generate code for (optional, one per line). By default all elements are typed.

**Names of types to create schema for:** Specifies the names of types to create a schema for (optional, one per line). If a type does not specify a namespace, the action matches all types in the assembly with the specified type. If a type specifies a namespace, only that type is matched. If a type ends with an asterisk character (\*), the tool matches all types that start with the string preceding the \*. If blank, the action generates schemas for all types in the assembly.

**Override executable filename:** Overrides the default executable (XSD.exe). If not specified, the 32-bit edition of Visual Build calls the 32-bit version of XSD.exe, and the 64-bit edition of Visual Build calls the 64-bit version of XSD.exe if available.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional).

## 3.10 Miscellaneous

### 3.10.1 Batch File

The Batch File action creates a step to run a batch file or command processor (cmd.exe) script. Visual Build starts and monitors the command processor and captures any output and logs it to the Output pane (and a log file if enabled).

#### Command Tab

#### Input Tab

#### Options Tab

## Advanced Tab

### Remote Tab

#### 3.10.1.1 Command Tab

This tab of the Batch File Command action specifies the batch file or command script to run.

**Command script:** Specifies the batch file or command script to invoke (required). The filename of a command script or batch file can be provided, or the contents of a script can be entered in the second field.

*Notes:*

- Single percent characters must be doubled (%%) to not be treated specially by Visual Build. Strings within percents (i.e., %PATH%) will also be expanded by Visual Build if not escaped.
- Bracket characters [ and ] normally denote a script expression to be inserted into a field; to insert literal brackets, use two bracket characters [ [ or ] ].
- The exit code of the command processor will be stored in the RUNPROGRAM\_EXITCODE temporary macro.
- To execute *Windows* shell (as opposed to command processor shell) commands (for instance, to open a document in its associated application, select `cmd /c call` in the processor field, `.cmd` in the file extension, and enter a command script of `start "shell command here"`.

#### 3.10.1.2 Options Tab

This tab of the Batch File Command action configures information about the command processor used to execute the batch file or command script.

**Start In:** The path that will be the starting directory for the process (optional).

**No echo:** Disables echoing of command output if checked.

**Disable command extensions:** Disables `cmd.exe` command extensions.

**Disable execution of AutoRun commands:** If checked, disables execution of AutoRun registry commands.

**Use old quote processing behavior:** If checked, turns on `cmd.exe /S` switch.

**Show application window:** Hides the console window when unchecked.

**Wait for Completion:** When checked, Visual Build waits for the step to finish before continuing and it displays any step output in the Output pane. Uncheck this item to immediately start the next step without waiting for the current step to complete. No output from that step will be shown in Visual Build and if the process is successfully started, it will be treated as a successful.

*Notes:*

- When this option is unchecked, the process ID of the launched process will be stored in the RUNPROGRAM\_PROCESSID temporary macro.
- When this option is checked, the exit code of the process will be stored in the RUNPROGRAM\_EXITCODE temporary macro.

**Success exit codes:** Visual Build determines the success of the process by examining the exit code

of the process. By default, a zero (0) exit code is considered successful, and any other code is a failure. Sometimes, an application will return non-zero exit codes to indicate partial success or additional information, and multiple ranges of success exit codes can be specified in the format `low1:hi1, low2:hi2, code3`. For instance, `0:5, 10` would cause the values 0 through 5 and 10 to be considered successful exit codes.

**File extension:** Specifies the file extension to use when creating a temporary file if script code is provided on the Commands tab (optional, defaults to `.bat`).

**Override default command interpreter executable:** Specifies the filename of the `cmd.exe` executable (optional, defaults to `cmd` if not provided).

*Note:* To call the 64-bit version of `cmd.exe` from the 32-bit edition of Visual Build, enter a value of `%WINDIR%\Sysnative\cmd.exe` in this field. To call the 32-bit version of `cmd.exe` from the 64-bit edition of Visual Build, enter a value of `%WINDIR%\SysWOW64\cmd.exe` in this field.

**Additional options:** Specifies additional flags for the command interpreter (optional).

### 3.10.2 Kill Process

The Kill Process action can be used to kill a Windows process, given its process ID or a Windows executable filename.

**Computer name:** Remote computer to kill the process on (optional, uses local computer if blank).

**Process ID or executable filename:** The process ID (base 10) or executable filename (i.e., `Notepad.exe`) of the process to kill. *Note:* `[Builder.ProcessID]` can be used to kill the current build process.

**Use wildcard search:** If unchecked, the executable filename must match exactly (non-case-sensitive). If checked, when searching on an executable filename, wildcards can be used for matching:

`[ ]` Any one character within the specified range (`[ a=f ]`) or set (`[ aef ]`).  
`^` Any one character not within the range (`[ ^a=f ]`) or set (`[ ^aef ]`).  
`%%` Any string of 0 (zero) or more characters (`%%Win%%`).  
`_` Any one character. A literal underscore must be escaped by placing inside brackets (`[ _ ]`).

When the step completes, the following temporary macros are created or updated:  
`KILLPROCESS_COUNT` = The number of processes that were terminated

*Note:* This action utilizes WMI, and your network configuration may need to be adjusted to allow WMI to operate through Windows Firewall; see this article for more details.

### 3.10.3 Play Sound

This custom action creates a step to play a sound within a build.

**Filename/sound:** The filename of sound file to play or the system sound to play (required).

**Wait for sound to finish playing:** If checked, the step will not complete until the sound has finished playing; if unchecked, the build will continue immediately after starting to play the sound.

**Repeat sound continuously:** If checked, the sound will be repeated continuously.

### 3.10.4 PowerShell

The PowerShell Command action creates a step to run a PowerShell script. Visual Build starts and monitors the PowerShell and captures any output and logs it to the Output pane (and a log file if enabled).

#### Command Tab

#### Input Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with PowerShell versions 1 thru 4 and may work with other versions as well.

#### 3.10.4.1 Command Tab

This tab of the PowerShell Action action configures information about the script to be run.

**Command script:** Specifies the script to invoke (required). The filename of a file can be provided, or the contents of a script can be entered in the second field.

#### Notes:

- Single percent characters must be doubled (%%) to not be treated specially by Visual Build. Strings within percent (i.e., %PATH%) will also be expanded by Visual Build if not escaped.
- Bracket characters [ and ] normally denote a script expression to be inserted into a field; to insert literal brackets, use two bracket characters [ [ or ] ].
- The exit code of the command processor will be stored in the RUNPROGRAM\_EXITCODE temporary macro.

Use the following code to read or write global macros from a PowerShell script (to transfer information between Visual Build and your script):

```
# create app object
$app = New-Object -com VisBuildSvr.Application

# retrieve global macros
$global = $app.Macros(2)

# READ A VALUE FROM Visual Build
# Note: values can also be passed to the script via the Parameters field on the
Input tab
# store a global macro value in a PowerShell variable
$val = $global.Item("ABC").Value

# echo the variable value (logged in Visual Build)
$val

# WRITE A VALUE TO Visual Build
# create/update macro with two values delimited by tab
$global.Add("XYZ", "val1`tval2")
```



```
# save changes to global macros (accessible in the following steps in Visual Build)
$gGlobal.Save()
```

### 3.10.4.2 Input Tab

This tab of the PowerShell and Batch File actions specifies any console inputs and parameters to pass to the script being run.

**Provide standard input from:** Specifies a file or string to provide to the program's standard input. This can be used for programs that read input from standard input when run in an automated fashion. If none is specified, no input will be sent to the program's standard input. Otherwise, the contents of the given file or the string entered will be passed to standard input when the program is run.

**Parameters:** Specifies parameter arguments to pass to the script or batch file. Within the script, the parameters will be available in the \$args variable:

```
#list all parameters
foreach ($arg in $args)
{
    $arg
}
```

### 3.10.4.3 Options Tab

This tab of the PowerShell Action action configures PowerShell options.

**Start In:** The path that will be the starting directory for the process (optional).

**Console file:** Loads the specified Windows PowerShell console file (optional).

**Input format:** Describes the format of data sent to PowerShell.

**Output format:** Determines how output from PowerShell is formatted.

**No logo:** Hides the copyright banner.

**No profile:** Does not use the user profile.

**Non-interactive:** Does not present an interactive prompt to the user.

**Execution policy:** Determines which PowerShell scripts (if any) will be allowed to run on your computer (default is Remote Signed):

- Default - The execution policy for the PowerShell environment will be used (as configured by the `Set-ExecutionPolicy` command). To enable PowerShell to execute local unsigned scripts, start PowerShell (running as admin) and enter the command `Set-ExecutionPolicy RemoteSigned`.
- Restricted - No scripts can be run. Windows PowerShell can be used only in interactive mode.
- All Signed - Only scripts signed by a trusted publisher can be run.
- Remote Signed - Downloaded scripts must be signed by a trusted publisher before they can be run. Use this option when using the Script field on the Command tab.
- Unrestricted - No restrictions; all Windows PowerShell scripts can be run. Use this option for when executing downloaded scripts.

**Show application window:** Hides the PowerShell console window when unchecked.

**Wait for Completion:** When checked, Visual Build waits for the step to finish before continuing and it displays any step output in the Output pane. Uncheck this item to immediately start the next step without waiting for the current step to complete. No output from that step will be shown in Visual Build and if the process is successfully started, it will be treated as a successful.

*Notes:*

- When this option is unchecked, the process ID of the launched process will be stored in the RUNPROGRAM\_PROCESSID temporary macro.
- When this option is checked, the exit code of the process will be stored in the RUNPROGRAM\_EXITCODE temporary macro.

**Success exit codes:** Visual Build determines the success of the process by examining the exit code of the process. By default, a zero (0) exit code is considered successful, and any other code is a failure. Sometimes, an application will return non-zero exit codes to indicate partial success or additional information, and multiple ranges of success exit codes can be specified in the format low1:hi1, low2:hi2, code3. For instance, 0:5, 10 would cause the values 0 through 5 and 10 to be considered successful exit codes.

**File extension:** Specifies the file extension to use when creating a temporary file if script code is provided on the Commands tab (optional, uses .ps1 if not specified).

**Version:** Starts the specified version of PowerShell (optional).

**Override default PowerShell executable:** Specifies the filename of the PowerShell executable (optional, defaults to powershell if not provided).

*Note:* To call the 64-bit version of PowerShell from the 32-bit edition of Visual Build, enter a value of %WINDIR%\Sysnative\WindowsPowerShell\v1.0\powershell.exe in this field. To call the 32-bit version of PowerShell from the 64-bit edition of Visual Build, enter a value of %WINDIR%\SysWOW64\WindowsPowerShell\v1.0\powershell.exe in this field.

**Additional options:** Specifies additional PowerShell flags (optional).

### 3.10.5 Read Registry

This action creates a step to read values from the Windows registry. The REG\_READ system macro can also be used to read values from the registry.

When the step completes, the following temporary macro is created or updated:  
READ\_REGISTRY\_VALUE = The registry value, value names, or subkey names that were read

**Root key:** The root registry key/hive to access.

**Subkey:** The subkey to access (for instance, Software\Microsoft\Windows\CurrentVersion), required.

**Value name:** The value to read (optional). The default value is read if an empty string is entered. To load the names of all values under the subkey, enter 5 asterisk characters (\*\*\*\*\*); to load the names of all subkeys under the subkey, enter 5 + characters (+++++). The Loop action can be used to process all matching value/key names.

**Fail if key or value doesn't exist:** If checked and the specified registry key or value doesn't exist, the step fails. If unchecked, the default value will be used.

**Default value:** The value to use if the registry key or value doesn't exist (optional).

**Macro name:** The name of the temporary macro in which to store the value that was read from the registry (optional, defaults to storing in the READ\_REG\_VALUE macro if not specified).

**Description:** The description of the temporary macro where the text is stored (optional).

**Do not escape special characters:** Does not escape (double) special characters in the registry when storing in a macro.

**Access 64-bit registry view:** On 64-bit Windows, this action in the 32-bit edition of Visual Build accesses the 32-bit registry view by default. If this option is checked, the action will access redirected keys in the 64-bit registry view. This option is only available in the 32-bit edition of Visual Build and has no effect on 32-bit Windows or shared registry keys in 64-bit Windows (for more information, see Registry Keys Affected by WOW64).

**Access 32-bit registry view:** On 64-bit Windows, this action in the 64-bit edition of Visual Build accesses the 64-bit registry view by default. If this option is checked, the action will access redirected keys in the 32-bit registry view. This option is only available in the 64-bit edition of Visual Build.

## Remote Tab

### 3.10.5.1 Remote Tab

This tab of the Read Registry and Write Registry actions is used to connect to a remote computer or specify alternate user credentials.

**Computer:** A remote computer to connect to (optional). If blank, the local computer's registry is modified.

*Note:* To access the registry on a remote computer, the Remote Procedure Call and Remote Registry services on the remote computer must be started. Your network configuration may also need to be adjusted to allow access through Windows Firewall. If the computer is joined to a workgroup and the *Force network logons using local accounts to authenticate as Guest* policy is enabled, this action will fail to connect to a remote computer's registry (this policy is enabled by default for a computer that is joined to a workgroup).

**Username:** Specifies the user account that will be used to connect to the registry (optional). If blank, the current user identity will be used. To specify a domain, use the format domain@username.

**Password:** Specifies the password of the user account (optional).

### 3.10.6 Set Current Dir

This custom action creates a step to set the current directory of the build process to the drive+path specified. This can be useful when specifying relative paths for filenames; all relative paths in fields of subsequent steps will be relative to the folder specified here.

*Note:* The current working directory can also be set automatically (to the path of the project file) via an application option.

**Directory:** The drive+path of the folder to make the new current directory.

### 3.10.7 Set Priority

The Set Priority action can be used to set the priority of the build process or another Windows process.

**Computer name:** Remote computer to set priority on (optional, uses local computer if blank).

**Process ID:** The process ID (base 10) of the process to set the priority on (required). Use [Builder.ProcessID] to set the process of the current build.

**Priority:** The priority to set the process to.

*Notes:*

- This action utilizes WMI, and your network configuration may need to be adjusted to allow WMI to operate through Windows Firewall; see this article for more details.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.10.8 Shut Down

The Shut Down action can be used to shut down, restart, power off, or log off Windows. This can be useful when performing builds in virtual machines, combined with automatic Windows logon.

**Computer name:** Remote computer to shut down (optional, shuts down local computer if blank).

**Operation:** The operation to perform (shut down, restart, power off, or log off).

**Force applications to close:** If checked, applications will be forced to close if they do not respond to the shut down request.

*Note:* This action utilizes WMI, and your network configuration may need to be adjusted to allow WMI to operate through Windows Firewall; see this article for more details.

### 3.10.9 Wait

The Wait action creates a step to pause the build until a given amount of time has elapsed, the given time or date/time is reached, until one or more files are created or modified, or until one or more processes exit or complete. This can be useful for recurring builds and parallel builds.

For instance, to build once every day, the entire build could be placed in a loop (by using the Repeat conditional build rule property) and the first step set to a Wait action that waits *Until a given date/time* of 14:30 is reached. To pause for 5 seconds, select *A given amount of time* and enter 00:00:05.

Or a build could be kicked off remotely by setting the action to wait for a file to be created or modified, and then creating or updating the file from a remote machine.

Another use for this step is for parallel builds. When multiple parts of a build which don't depend on each other need to be run simultaneously, the master build project can launch each parallel part of the build from a VisBuildPro Project step marked to not wait for completion. If the rest of the build is dependent on the completion of the parallel builds, the build could be set up as demonstrated in the Chain.bld sample.

*Note:* The Sleep method can be used to pause from script code.

**Wait until:** Specifies what to wait for.

**Time / Date+time / Filename(s) / Process ID(s):** The amount of time to wait, the time or date+time to wait for, one or more filenames to wait to be created or modified, or one or more process IDs to wait for. Elapsed time should be entered in the form HH:MM:SS (hours, minutes, seconds), MM:SS (minutes, seconds) or SS (seconds); a given time should be entered in military format in the form HH:MM, date+time should be entered in the form MM/DD/YYYY HH:MM, and a process ID should be entered in decimal (base 10) format.

**Delete files before waiting:** If checked, any existing files to be wait on will be deleted before waiting (applies only to file option).

**Wait for all files to be created or modified or all processes to exit:** If checked, the action will wait for all listed files to be created or modified or all processes to exit. If unchecked, the action will continue on the first created or modified file or completed process.

**Don't wait if file already exists:** When waiting until a file is created, if this checkbox is checked, the step will stop waiting if the specified file has already been created by the time the step starts (applies only to file option).

**Success string or codes:** If provided, when waiting for files, the first line of each created or modified file will be searched for the specified string, and if the string is not found, the action will fail. If waiting for processes to complete, the exit code of the process will be compared with the exit codes in this field to determine success or failure. By default, a zero (0) exit code is considered successful, and any other code is a failure. Sometimes, an application will return non-zero exit codes to indicate partial success or additional information, and multiple ranges of success exit codes can be specified in the format low1:hi1, low2:hi2, code3. For instance, 0:5, 10 would cause the values 0 through 5 and 10 to be considered successful exit codes.

### 3.10.10 Write Registry

This action creates a step to create, update, or delete registry values. Use the Read Registry action to read a value from the registry.

**Root key:** The root registry key/hive to access.

**Subkey:** The subkey to access (for instance, `Software\Microsoft\Windows\CurrentVersion`), required. Any subkeys that don't exist will be created when creating values.

**Value name:** The value to create/update or delete (optional). The default value is updated if an empty string is entered. To delete the subkey and all child keys/values, enter 5 asterisk characters, \*\*\*\*\* ( **Important:** be careful with this, as all keys and values below the subkey will be recursively deleted).

**Delete the value:** If checked, the registry key or value is deleted if it exists.

**Value type:** Type of value to create (string or DWORD).

**Value data:** The data to write to the registry value (required unless deleting).

**Access 64-bit registry view:** On 64-bit Windows, this action in the 32-bit edition of Visual Build accesses the 32-bit registry view by default. If this option is checked, the action will access redirected keys in the 64-bit registry view. This option is only available in the 32-bit edition of Visual Build and has no effect on 32-bit Windows or shared registry keys in 64-bit Windows (for more information, see Registry Keys Affected by WOW64).

**Access 32-bit registry view:** On 64-bit Windows, this action in the 64-bit edition of Visual Build accesses the 64-bit registry view by default. If this option is checked, the action will access redirected keys in the 32-bit registry view. This option is only available in the 64-bit edition of Visual Build.

## Remote Tab

### 3.11 Network

#### 3.11.1 FTP

The FTP action creates a step to copy, move, delete, list, or synchronize files and folders to or from an FTP, FTPS or SFTP server. It supports secure transfers, processing of subdirectories, incremental copying (copying only files that have changed since previous copy), synchronization of folders (by purging extra files from the destination folder), deleting of files, including and excluding files and folders via file masks, configurable logging of operations, a debug mode to only display the files that would be processed, and more.

When the step completes, the following temporary macros are created or updated:  
 FTP\_TRANSFER\_COUNT = The number of files that were successfully transferred or listed  
 FTP\_TRANSFER\_SIZE = The total size (in bytes) of all transferred or listed files  
 FTP\_DELETE\_COUNT = The number of files deleted when synchronizing  
 FTP\_HOST\_FINGERPRINT = The host key fingerprint (for SFTP connections only)

## Server Tab

## Transfer Tab

## Security Tab

## Options Tab

*Note:* See the Network sample for an example of the FTP action.

### 3.11.1.1 Server Tab

This tab of the FTP action specifies server and login settings.

**FTP server:** The server name (i.e., ftp.kinook.com) or IP address (required).

**Port:** The port to connect on (optional, default is 21 for non-secure connections, 22 for SFTP and 990 for SSL).

**Timeout:** How long (in seconds) to wait for a connection or response before timing out (required). Enter 0 to wait indefinitely.

**Username:** The username to login as (required).

**Password:** User's password (optional).

**Use passive mode:** When the client is in active mode (unchecked), the remote server establishes a connection with the client to transfer data. When the client is in passive mode (checked), the client is responsible for establishing the connection. This may be necessary when accessing some servers through a proxy server or those that are protected behind a firewall.

**Priority:** Specifies the priority for file transfers:

Default/No Balances resource utilization and transfer speed.

rma

Background Significantly reduces the memory, processor and network resource utilization for the transfer. It is typically used with worker threads running in the background when the amount of time required perform the transfer is not critical.

Low Lowers the overall resource utilization for the transfer and meters the bandwidth allocated for the transfer. This priority will increase the average amount of time required to complete a file transfer.

High Increases the overall resource utilization for the transfer, allocating more memory for internal buffering. It can be used when it is important to transfer the file quickly, and there are no other threads currently performing file transfers at the time.

Critical Can significantly increase processor, memory and network utilization while attempting to transfer the file as quickly as possible. No progress events will be logged during the transfer.

**Proxy type:** Configures the type of proxy server (if any) to connect to:

- **None:** No proxy server.
- **User:** Specifies that the client is not logged into the proxy server. The USER command is sent in the format `username@ftpsite` followed by the password. This is the format used with the Gauntlet proxy server.
- **Login:** Specifies that the client is logged into the proxy server. The USER command is then sent in the format `username@ftpsite` followed by the password. This is the format used by the InterLock proxy server.
- **Open:** Specifies that the client is not logged into the proxy server. The OPEN command is sent specifying the host name, followed by the username and password.
- **Site:** Specifies that the client is logged into the server. The SITE command is sent, specifying the host name, followed by the username and the password.

**Proxy server:** The proxy server name (i.e., `ftp.kinook.com`) or IP address (required).

**Proxy port:** The port to connect on (required).

**Proxy username:** The username to login as (optional).

**Proxy password:** User's password (optional).

*Note:* Proxy settings do not apply for SFTP connections.

### 3.11.1.2 Transfer Tab

This tab of the FTP action specifies file transfer information.

**Action:** The action to perform (Put [upload], Get [download], List [dir], or Delete).

**Transfer type:** Whether to transfer as ASCII or binary. Use ASCII transfer only when transferring between Windows and Unix systems to convert line endings between CR/LF and CR. Use binary transfer for all other transfers (do not use ASCII for files containing non-text data as this can corrupt the destination file).

*Note:* Transfer type does not apply for SFTP transfers and will be ignored.

**Retries:** The number of times to retry the FTP operation if a retryable timeout occurs while transferring a file (0 or blank for no retries).

**Local folder:** The local folder to transfer from when putting or to transfer to when getting (required for Get and Put actions).

*Note:* For backward compatibility, this can also be a local path+filename or mask to upload when putting (if the filename contains wildcards [`?` or `*` characters], all files matching the wildcard in the specified directory will be uploaded).

**Remote folder:** Path on the server to transfer the file to when putting or to transfer to when getting (optional [defaults to the home directory if not specified]).

*Note:* Some FTP servers (for instance, VAX/VMS) may require the *Do not use directory as reported by server* option on the Server tab to be checked.

**Files to process:** Used to specify files or folders to specifically include or exclude (optional).

**Include subdirectories (recursive):** Determines if subdirectories will be searched recursively for copying, synchronizing, and deleting.

**Copy only files that have changed (incremental copy):** Enables incremental copying of files.

Before copying a file, if this option is checked, the action compares the size (unless performing an ASCII transfer) and timestamp of the source and destination files and only copies the files if the file does not exist in the destination folder or the size or timestamps are different. Any files that are already up-to-date in the destination will not be copied. This option also takes into account daylight saving time differences (files that are exactly one hour difference will not be copied).

*Note:* The FTP server must support the SIZE command to retrieve file sizes and the MDTM command to retrieve and set file timestamps in order to perform incremental copying.

**Delete destination files and folders that no longer exist (synchronize):** If checked, any files or subfolders in the destination folder that do not exist in the source folder will be deleted from the destination folder. Use in conjunction with incremental copy to synchronize the contents of two folders. The Include/Exclude fields are also used to include or exclude files from the purge operation. For the Delete action, check this option to also delete empty subfolders.

**Delete empty root destination folder:** If checked, the root destination folder will also be deleted if all files were deleted when synchronizing. For the Delete action, checking this option indicates that the root folder should be deleted too if all files and subfolders were deleted.

*Note:* Be very careful when using this option, as many files can be quickly deleted when this option is specified. It is advisable to use the *Do not copy* option when testing a synchronize operation to show which files would be removed.

**Do not copy:** If checked, no files or folders will be copied, synchronized, or deleted, but any enabled logging will be displayed and a count of files that would have been copied, synchronized, or deleted will be logged. Useful for debugging purposes.

**Logging:** Level of logging to perform on network communications or file transfers:

- None: No logging (other than a count of files transferred)
- All: All network communications logged
- Errors: Only errors logged
- Errors/warnings: Any errors and warnings logged
- All + Hex dump: Logs all networks communications and a hex dump of all data transferred
- Filenames: The filename of each file processed is logged
- Filenames + Progress: Logs each file processed and logs the progress while transferring files

**Log every x percent:** If Logging is set to Filenames + Progress and a value between 1 and 99 is entered in this field, a progress update will be logged every n percent of completion for each file. If the field is empty or file sizes are not available, a progress update will be logged every 5 seconds.

*Note:* The FTP action will create the destination folder for a file if it doesn't already exist.

### 3.11.1.3 Security Tab

This tab of the FTP action specifies security settings for secure FTP connections.

**Protocol:** Security protocol to use when connecting to the remote server:

- None: No security protocol will be used
- SSL, TLS, or PCT: Either the SSL 2.0, SSL 3.0, PCT or TLS protocols will be used when establishing a secure connection. The correct protocol is automatically selected.
- SSL: Either SSL 2.0 or SSL 3.0 may be used when establishing a secure connection. The correct protocol is automatically selected.
- TLS: The TLS 1.0 protocol will be used when establishing a secure connection.
- PCT: The PCT 1.0 protocol will be used when establishing a secure connection.
- SSH: Either SSH 1.0 or SSH 2.0 will be used when establishing a secure connection, based on the version of the protocol that is supported by the server, and the SFTP protocol will be used for transferring files.



- **SSH v1:** The SSH 1.0 protocol will be used when establishing the connection, and the SFTP protocol will be used for transferring files. This is an older version of the protocol which should not be used unless explicitly required by the server. Most modern SFTP server support version 2.0 of the protocol.
- **SSH v2:** The SSH 2.0 protocol should be used when establishing the connection, and the SFTP protocol will be used for transferring files. This is the default version of the protocol that is supported by most SFTP servers.

**Options:** Options related to the security protocol (does not apply for SFTP):

- **Default:** specifies that the client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL, PCT or TLS protocols.
- **Explicit SSL:** The client will first send an AUTH TLS command to the server. If the server does not accept this command, it will then send an AUTH SSL command. If both commands are rejected by the server, an explicit SSL session cannot be established.
- **Implicit SSL:** Negotiates a secure session as soon as the connection is established and does not require a command.

### Certificate Info

**Location:** The location to retrieve the certificate from (user store, machine store, or PFX file).

**Store name/file:** The name of the store to open or the PFX file to retrieve the certificate from (optional).

**Certificate name:** Friendly name of the certificate to use (optional). Note that the name must match completely, but the comparison is not case sensitive. If no matching certificate is found, the action will then attempt to find a certificate that has a matching common name (also called the certificate subject). This comparison is less stringent, and the first partial match will be returned. If this second search fails, the action will return an error indicating that the certificate could not be found.

**Password:** The certificate password (applies only for certificates in a PFX file).

*Note:* Certificate info does not apply for SSH/SFTP.

**Send commands unencrypted:** If checked, the command channel used to send commands to the server and receive command result and status information from the server will not be encrypted. This may be necessary to allow a secure FTP connection through a firewall. Changing the mode for the data channel requires that the server support the PROT command. If this command is not supported by the server, the function will fail and the channel mode will remain unchanged.

**Transfer data unencrypted:** If checked, the channel used to transfer data with the server will not be encrypted. Changing the mode for the data channel requires that the server support the PROT command. If this command is not supported by the server, the function will fail and the channel mode will remain unchanged.

**Verify host fingerprint:** If checked, the specified fingerprint will be compared with the fingerprint value returned by the server, and the step will fail if they do not match. Build the step once to determine the host key, then use the value from the build output here for future use (to prevent man-in-the-middle security attacks). This option applies only for SSH/SFTP connections.

#### 3.11.1.4 Options Tab

This tab of the FTP action specifies additional options.

**Verify transfers:** This option specifies that file transfers should be automatically verified after the transfer has completed. If the server supports the XMD5 command, the transfer will be verified by calculating an MD5 hash of the file contents. If the server does not support the XMD5 command, but

does support the XCRC command, the transfer will be verified by calculating a CRC32 checksum of the file contents. If neither the XMD5 or XCRC commands are supported, the transfer is verified by comparing the size of the file. Automatic file verification is only performed for binary mode transfers because of the end-of-line conversion that may occur when text files are uploaded or downloaded.

**No authorization:** This option specifies that the server does not require authentication or that it requires an alternate authentication method. When this option is used, the client connection is flagged as authenticated as soon as the connection to the server has been established. Note that using this option to bypass authentication may result in subsequent errors when attempting to retrieve a directory listing or transfer a file. It is recommended that you consult the technical reference documentation for the server to determine its specific authentication requirements.

**Attempt to keep the connection with the server active:** This option specifies that the client should attempt to keep the connection with the server active for an extended period of time.

**Virtual hosting:** This option specifies that the server supports virtual hosting, where multiple domains are hosted by a server using the same external IP address. If this option is enabled, the client will send the HOST command to the server upon establishing a connection.

**Tunneled connection:** This option specifies that a tunneled TCP connection and/or port-forwarding is being used to establish the connection to the server. This changes the behavior of the client with regards to internal checks of the destination IP address and remote port number, default feature selection and how the connection is established. This option also forces all connections to be outbound and enables the firewall compatibility features in the client.

**Trusted site:** This option specifies that the server is trusted. The server certificate will not be validated and the connection will always be permitted. This option only affects connections using either the SSL or TLS protocols.

**Do not use directory as reported by server:** If unchecked (the default), the action will retrieve the full current directory name from the server and use that to set the current directory when processing subfolders (this may be necessary if the Remote folder is specified as a relative path). If checked, the action will use the path exactly as provided by the user on the Transfer tab (this may be necessary if the server returns paths with non-standard notation; i.e., `:[x.y]` rather than `/x/y` as used by some VAX/VMS FTP servers, and absolute paths in slash notation should be specified in the Remote folder field in this case).

### 3.11.2 HTTP

The HTTP action creates a step to execute commands on an HTTP server. It supports secure connections, get, put, post, and delete requests, and more.

#### Server Tab

#### Command Tab

#### Options Tab

#### Security Tab

#### Form Tab

*Note:* See the Network sample for an example of the HTTP action.

### 3.11.2.1 Server Tab

This tab of the HTTP action specifies server and login settings.

**HTTP server:** The server name (i.e., www.kinook.com) or IP address (required).

**Port:** The port to connect on (optional, default is 80 for unsecured or 443 for secured connections).

**Timeout:** How long (in seconds) to wait for a connection or response before timing out (required). Enter 0 to wait indefinitely.

**Username:** The username to login as (optional).

**Password:** User's password (optional).

**HTTP version:** The HTTP version to use when sending requests to the server (required, defaults to 1.1).

**Encoding:** Specifies the type of encoding to be applied to the content of a HTTP request.

- **None:** No encoding will be applied to the content of a request, and no Content-Type header will be generated.
- **URL:** URL encoding will be applied to the content of a request, and a Content-Type header will be generated with the value "application/x-www-form-urlencoded"
- **XML:** URL encoding will be applied to the content of a request, EXCEPT that spaces will not be replaced by +. This encoding type is intended for use with XML parsers that do not recognize + as a space. A Content-Type header will be generated with the value "application/x-www-form-urlencoded".

**Priority:** Specifies the priority for file transfers:

Default/No Balances resource utilization and transfer speed.

normal

**Background** Significantly reduces the memory, processor and network resource utilization for the transfer. It is typically used with worker threads running in the background when the amount of time required perform the transfer is not critical.

**Low** Lowers the overall resource utilization for the transfer and meters the bandwidth allocated for the transfer. This priority will increase the average amount of time required to complete a file transfer.

**High** Increases the overall resource utilization for the transfer, allocating more memory for internal buffering. It can be used when it is important to transfer the file quickly, and there are no other threads currently performing file transfers at the time.

**Critical** Can significantly increase processor, memory and network utilization while attempting to transfer the file as quickly as possible. No progress events will be logged during the transfer.

**Proxy type:** Configures the type of HTTP proxy server (if any) to connect to:

- **None:** No proxy server.
- **CERN:** The client is connecting to a CERN proxy server, and all resource requests will be specified using a complete URL. This proxy type should be used with standard connections.
- **Tunnel:** The client is connecting through a standard, non-secure CERN proxy server to a secure server. A protocol must also be selected on the Security tab.
- **Windows Config:** The configuration options for the current system should be used. These settings are defined through the Internet Options in the control panel and are the same proxy server settings used by Internet Explorer.

**Proxy server:** The proxy server name (i.e., kinook.com) or IP address (required).

**Proxy port:** The port to connect on (required).

**Proxy username:** The username to login as (optional).

**Proxy password:** User's password (optional).

### 3.11.2.2 Command Tab

This tab of the HTTP action specifies the HTTP request information.

**Put:** The put command to perform (*None*, *Put file*, *Post file*, *Post form data*, or *Delete*). For the *Post form data* command, enter the form information on the Form tab.

**Get:** The get command to perform (*Get to file*, *Get to macro*, or *None*).

**URL:** The URL on the server to submit the request to (required). Do not include the protocol or domain name (i.e., `http://www.site.com`), but only the local resource URL itself (i.e., `/path/to/file.html`).

**Put file:** For *Put file* or *Post file* commands, the name of the file to put (required).

**Get file:** For the *Get to file* command, the name of the local file to get to (required).

**Macro:** For the *Get to macro* command, the name of a temporary macro to store the response in (required).

**Do not escape special characters:** Does not escape (double) special characters in the response when storing in a macro.

**Transfer type:** The transfer mode:

- Binary: the resource data is copied to the local system exactly as it is stored on the server.
- ASCII: If the resource being downloaded from the server is textual, the data is automatically converted so that the end of line character sequence is compatible with the Windows platform. Individual carriage return or linefeed characters are converted to carriage return/linefeed character sequences.

**Enable compression:** Enables support for data compression. This option indicates to the server whether or not it is acceptable to compress the data that is returned to the client. If compression is enabled, the client will advertise that it will accept compressed data by setting the Accept-Encoding request header. The server will decide whether a resource being requested can be compressed. If the data is compressed, the library will automatically expand the data before returning it to the caller. Enabling compression does not guarantee that the data returned by the server will actually be compressed, it only informs the server that the client is willing to accept compressed data. Whether or not a particular resource is compressed depends on the server configuration, and the server may decide to only compress certain types of resources, such as text files.

**Form data:** For the *Post form data* command, enter the form data to be posted; for the *Post file* command, enter the name of the form field that the script expects (if empty, a default field name of "File1" is used).

**Logging:** Level of logging to perform on network communications or file transfers:

- None: No logging (other than a count of files transferred)
- All: All network communications logged
- Errors: Only errors logged
- Errors/warnings: Any errors and warnings logged
- All + Hex dump: Logs all networks communications and a hex dump of all data transferred
- Progress: Logs progress while transferring file

- **Data:** Logs data that is returned by the server

**Log every x percent:** If Logging is set to Progress and a value between 1 and 99 is entered in this field, a progress update will be logged every n percent of completion when transferring a file. If the field is empty or file sizes are not available, a progress update will be logged every 5 seconds.

*Note:* The HTTP action will create the destination folder for a local file if it doesn't already exist.

### 3.11.2.3 Options Tab

This tab of the HTTP action specifies the HTTP request header and cookie.

**Headers:** Enter a request header name and value and click Insert to add to the list. Select an item in the list and update the user names and click Update to update, or click Delete to remove it from the list.

**Cookies:** Enter a cookie name and value and click Insert to add to the list. Select an item in the list and update the user names and click Update to update, or click Delete to remove it from the list.

### 3.11.2.4 Security Tab

This tab of the HTTP action specifies security settings for secure HTTP connections.

**Protocol:** Security protocol to use when connecting to the remote server:

- **None:** No security protocol will be used
- **SSL, TLS, or PCT:** Either the SSL 2.0, SSL 3.0, PCT or TLS protocols will be used when establishing a secure connection. The correct protocol is automatically selected.
- **SSL:** Either SSL 2.0 or SSL 3.0 may be used when establishing a secure connection. The correct protocol is automatically selected.
- **TLS:** The TLS 1.0 protocol will be used when establishing a secure connection.
- **PCT:** The PCT 1.0 protocol will be used when establishing a secure connection.

#### Certificate Info

**Location:** The location to retrieve the certificate from (user store, machine store, or PFX file).

**Store name/file:** The name of the store to open or the PFX file to retrieve the certificate from (optional).

**Certificate name:** Friendly name of the certificate to use (optional). Note that the name must match completely, but the comparison is not case sensitive. If no matching certificate is found, the action will then attempt to find a certificate that has a matching common name (also called the certificate subject). This comparison is less stringent, and the first partial match will be returned. If this second search fails, the action will return an error indicating that the certificate could not be found.

**Password:** The certificate password (applies only for certificates in a PFX file).

### 3.11.2.5 Form Tab

This tab of the HTTP action specifies the form information. Form data can be entered in the grid or the raw form data field (but not both).

**Form fields:** Enter a form field name and value and click Insert to add to the list. Select an item in the list and update the user names and click Update to update, or click Delete to remove it from the list.

**Raw form data:** For the *Post form data* command, enter the raw form data to be posted (if not specifying form fields above) in the format `field1=value1&field2=value2`; for the *Post file* command,

enter the name of the form field that the script expects (if empty, a default field name of *File1* is used).

### 3.11.3 Map Drive

The Map Drive action can be used to connect, disconnect or list mapped network paths and to substitute, delete, or list substituted drives.

**Operation:** The operation to perform.

**Drive letter:** The drive letter to assign or delete (optional when assigning or mapping drives).

If not specified, the next available drive letter is used and stored in the **MAPDRIVE\_LETTER** temporary macro.

**Folder or network path:** The local drive and folder or network path to map to a drive letter (required).

*Note:* The following fields apply only when mapping a network path

**Reconnect at login:** Determines if the drive letter mapping is persisted between login sessions (applies only to Map operation).

**Force disconnect:** If checked, forces disconnection even if there are open files or searches pending on the connection (applies only to Disconnect operation).

**Username:** The domain\username credentials to use (optional).

**Password:** The password to provide for credentials (optional).

**Show command-line:** Determines if the command-line used is logged.

### 3.11.4 Newsgroup Post

The Newsgroup Post action creates a step to post a message to one or more newsgroup servers.

#### Server tab

#### Message tab

#### Security tab

*Notes:*

- See the Network sample for an example of the Newsgroup Post action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.11.4.1 Server Tab

This tab of the Newsgroup Post action specifies server and login settings.

**News Server:** The NNTP server name or IP address (required).

**Port:** The port to connect on (optional, default is 119 for unsecured and 563 for secured connections).

**Timeout:** How long (in seconds) to wait for a connection or response before timing out (required). Enter 0 to wait indefinitely.

**Username:** Username for login to NNTP server (optional).

**Password:** Password for login (optional).

**Priority:** Specifies the message priority.

**Logging:** Level of logging to perform on network communications (none, all, errors, errors/warning, all+hex dump).

#### 3.11.4.2 Message Tab

This tab of the Newsgroup Post action configures information about the message being sent.

**From:** The from email address to use on the message (optional). To include a name with the address, use the format "Person's Name" <from@company.com>.

**Newsgroups:** Newsgroups to post the message to; separate multiple newsgroups with a comma (required).

**Subject:** Subject of the message (optional).

**Message:** Text of the message (optional).

**Send as HTML:** If checked (the Message field should contain HTML), the message body is marked as HTML instead of plain text.

**Attachments:** List of files to attach to the message, each on its own line (optional). If XML logging is used and you wish to send the log file as an attachment from a within the build, the closing document tags must first be added to the log file so it will be a valid XML document (they aren't added by Visual Build until after the last step completes). This is demonstrated in the Logging.bld sample.

#### 3.11.4.3 Security Tab

This tab of the Newsgroup Post action specifies security settings for secure SMTP connections.

**Protocol:** Security protocol to use when connecting to the remote server:

- None: No security protocol will be used
- SSL, TLS, or PCT: Either the SSL 2.0, SSL 3.0, PCT or TLS protocols will be used when establishing a secure connection. The correct protocol is automatically selected.
- SSL: Either SSL 2.0 or SSL 3.0 may be used when establishing a secure connection. The correct protocol is automatically selected.
- TLS: The TLS 1.0 protocol will be used when establishing a secure connection.
- PCT: The PCT 1.0 protocol will be used when establishing a secure connection.

##### Certificate Info

**Location:** The location to retrieve the certificate from (user store, machine store, or PFX file).

**Store name/file:** The name of the store to open or the PFX file to retrieve the certificate from (optional).

**Certificate name:** Friendly name of the certificate to use (optional). Note that the name must match completely, but the comparison is not case sensitive. If no matching certificate is found, the action will then attempt to find a certificate that has a matching common name (also called the certificate subject). This comparison is less stringent, and the first partial match will be returned. If this second search fails, the action will return an error indicating that the certificate could not be found.

**Password:** The certificate password (applies only for certificates in a PFX file).

### 3.11.5 PLink Tunnel

The PLink Tunnel action creates an SSH tunnel for secure FTP, Telnet, and SMTP authentication over SSH.

*Notes:*

- Use the Telnet action with SSH security protocol for secure SSH connections, and the FTP action with SFTP security for secure SFTP file transfers.
- [plink.exe](#) must be available in the PATH environment variable, which this action uses to set up the tunnel.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

The process ID of the PLink process will be stored in the RUNPROGRAM\_PROCESSID temporary macro. This can be used in a Kill Process action after using the tunnel to tear down the tunnel.

You must first use [putty.exe](#) to set up the keys needed for secure access. In the Putty application, configure a connection to the host, and open it once to accept the RSA/DSA key provided by SSH on that host. You are typically prompted when connecting to accept the key; answering Yes will add the key to your registry, enabling future access without prompting. You can also create a saved session in Putty, then specify this session name in the PLink Tunnel step instead of the individual properties. This session can also specify a private key file, eliminating the need to manually define each tunnel property or the configuration step above. See the PuTTY web site for more information.

You will also be prompted for the username and password for each protocol. Alternatively, you can utilize public key authentication to avoid having to specify a username and password in the build script -- see the PuTTY [user manual](#) for more details on setting this up.

### 3.11.6 Send Mail

The Send Mail action creates a step to send email to one or more people. This can be used, for instance, to send email on a successful build or when a build fails.

**Server tab**

**Message tab**

**Security tab**

*Note:* See the Network sample for an example of the Send Mail action.

#### 3.11.6.1 Server Tab

This tab of the Send Mail action specifies server and login settings.

**Mail Server:** The SMTP server name or IP address (required).

**Port:** The port to connect on (optional, default is 25 for unsecured and 465 for secured connections).

**Timeout:** How long (in seconds) to wait for a connection or response before timing out (required). Enter 0 to wait indefinitely.

**Username:** Username for login to SMTP server (optional). When authenticating, SMTP requires that



the server support the extended SMTP protocol and the AUTH LOGIN command. If your server doesn't support that, use an empty Username field, which will cause the action to not use extended SMTP or attempt to authenticate a user.

**Password:** Password for login (optional).

**Domain:** Local domain name for client (optional).

**Priority:** Specifies the message priority.

**Return a message on success, failure, delay:** If checked, these items will cause a message to be returned to the sender on success, failure, or delay in sending a message.

*Note:* The above delivery options are only available on those mail servers which support delivery status notification (DSN) using the extended SMTP protocol.

**Return entire message:** Returns the entire message instead of a simple notification for any of the above options.

**Logging:** Level of logging to perform on network communications (none, all, errors, errors/warning, all+hex dump).

**POP-before-SMTP authentication:** If checked, authenticates against the specified POP server before sending the message (required by some mail servers before sending via SMTP), and the remaining POP fields must be entered.

**POP Server:** The POP server name or IP address.

**POP Port:** The POP port to connect on (optional, default is 110 for unsecured and 563 for secured connections).

**POP Username:** Username for login to POP server.

**POP Password:** Password for POP authenticate.

### 3.11.6.2 Message Tab

This tab of the Send Mail action configures information about the message being sent.

**From:** The from email address to use on the message (optional). To include a name with the address, use the format "Person's Name" <from@company.com>.

**To:** Email address to send the message to; separate multiple addresses with a comma (required). To include a name with an address, use the format "Person's Name" <to@company.com>.

**Cc:** Additional email addresses to copy the message to; separate multiple addresses with a comma (optional). To include a name with an address, use the format "Person's Name" <to@company.com>.

**Bcc:** Additional email address to blind copy the message to; separate multiple addresses with a comma (optional). To include a name with an address, use the format "Person's Name" <to@company.com>.

**Subject:** Subject of the message (optional).

**Message:** Text of the message (optional).

**Send as HTML:** If checked (the Message field should contain HTML), the message body is marked as HTML instead of plain text.

**Attachments:** List of files to attach to the message, each on its own line (optional). If XML logging is used and you wish to send the log file as an attachment from a within the build, the closing document tags must first be added to the log file so it will be a valid XML document (they aren't added by Visual Build until after the last step completes). This is demonstrated in the Logging.bld sample.

### 3.11.6.3 Security Tab

This tab of the Send Mail action specifies security settings for secure SMTP and POP connections.

**Protocol:** Security protocol to use when connecting to the remote server:

- None: No security protocol will be used
- SSL, TLS, or PCT: Either the SSL 2.0, SSL 3.0, PCT or TLS protocols will be used when establishing a secure connection. The correct protocol is automatically selected.
- SSL: Either SSL 2.0 or SSL 3.0 may be used when establishing a secure connection. The correct protocol is automatically selected.
- TLS: The TLS 1.0 protocol will be used when establishing a secure connection.
- PCT: The PCT 1.0 protocol will be used when establishing a secure connection.

**Options:** Options related to the security protocol:

- Default: The client should attempt to establish a secure connection with the server. Note that the server must support secure connections using either the SSL, PCT or TLS protocols.
- Implicit SSL: Select when the server expects an implicit SSL connection.
- Explicit SSL: Select when the server expects an explicit SSL connection.

#### Certificate Info

**Location:** The location to retrieve the certificate from (user store, machine store, or PFX file).

**Store name/file:** The name of the store to open or the PFX file to retrieve the certificate from (optional).

**Certificate name:** Friendly name of the certificate to use (optional). Note that the name must match completely, but the comparison is not case sensitive. If no matching certificate is found, the action will then attempt to find a certificate that has a matching common name (also called the certificate subject). This comparison is less stringent, and the first partial match will be returned. If this second search fails, the action will return an error indicating that the certificate could not be found.

**Password:** The certificate password (applies only for certificates in a PFX file).

*Note:* The Plink Tunnel action can be used for creating a secure SSH tunnel for use with the SMTP action if your server does not support SSL (only authentication, and not data transfer, is encrypted when using SMTP over SSH).

### 3.11.7 Telnet

The Telnet action creates a step to execute a script on a remote computer that supports Telnet or SSH sessions.

When the step completes, the following temporary macros are created or updated:

- **TELNET\_HOST\_FINGERPRINT** = The host key fingerprint (for SSH connections only)

#### Telnet tab

## Script tab

### Security tab

*Notes:*

- See the Network sample for an example of the Telnet action.
- RSH.exe can also be used from the Run Program action to execute a command on a remote computer running the Remote Shell service. Parameters can be passed to the remote command or script, or the FTP action can be used to send a custom script and then execute it remotely.

#### 3.11.7.1 Telnet Tab

This tab of the Telnet action specifies server information.

**Server:** The Telnet server name or IP address (required).

**Port:** The port to connect on (options, default is 23 for Telnet, 22 for SSH, and 992 for SSL).

**Timeout:** How long (in seconds) to wait for a connection or response before timing out (required). Enter 0 to wait indefinitely.

**Delay:** Specifies an amount of time (in milliseconds) to wait for a server response for each sent command. If 0 (the default), the action will wait until there is nothing to read from the server.

**Username:** User to login as (optional). If not provided, the script must manually login.

**Password:** Password for user logging in (optional).

*Note:* If a username and password are provided, the action will attempt to login to the remote server automatically. It is specifically designed to work with most UNIX based servers, and may work with other servers that use a similar login process. It scans the response text for a username prompt and then replies with the specified username. If that is successful, it will then scan for a password prompt and provide the specified password. The next response to be checked by the first Script response match will return any welcome message from the server (this is typically also followed by a command prompt). Because it is designed for UNIX based systems, it may not work with servers running on other operating system platforms such as Windows or VMS. In this case, leave the username and password fields blank and provide the necessary login commands in the Script field.

**Proxy type:** Configures the type of proxy server (if any) to connect to:

- None: No proxy server.
- HTTP: Specifies that the client is not logged into the proxy server. The USER command is sent in the format username@ftpsite followed by the password. This is the format used with the Gauntlet proxy server.
- Telnet: Specifies that the client is logged into the proxy server. The USER command is then sent in the format username@ftpsite followed by the password. This is the format used by the InterLock proxy server.

**Proxy server:** The proxy server name or IP address (required).

**Proxy port:** The port to connect on (required).

**Proxy username:** The username to login as (optional).

**Proxy password:** User's password (optional).

*Note:* Proxy settings only apply for SSH connections.

### 3.11.7.2 Script Tab

This tab of the Telnet action specifies the script to execute on the server.

**Delay:** A delay (in milliseconds) to wait for a server response for each command that is sent (required).

**Script:** Specifies responses from the server to look for and commands to send to the server, each on a separate line (required). Indicate an initial response string to expect from the server, followed by a command to send, a response to expect, and so on, in the form:

```
initial server response to look for<<optional custom delay>>
first command to send
server response to look for<<optional custom delay>>
second command to send
server response to look for<<optional custom delay>>
...
```

If the response string to match is found anywhere in the server response, the next send string will be sent until the entire script has been performed. If the response string is not found, an error occurs. To perform a negative match (anything except the response string), prefix the response string with an exclamation point (!).

After connecting, the Telnet action will process all text sent by the server, looking for a match with the response text provided in the Script field. If a match is found within the specified Delay (or if a "not" match [line prefixed with !] is not found within the delay time), the command from the next line of the script will be sent. To customize the delay for each response, add the custom delay amount at the end of the line within << >> characters (e.g., <<5000>>). If a match is not found, the step will fail. This will be repeated for each pair of lines in the script.

*Note:* The last line of the script should be the final response to expect from the server for the command on the previous line.

**Logging:** Level of logging to perform on network communications (none, all, errors, errors/warning, all+hex dump, or text). Text (the default) logs all text sent and received to/from the server, which can be useful for debugging purposes.

### 3.11.7.3 Security Tab

This tab of the Telnet action specifies security settings for secure Telnet connections.

**Protocol:** Security protocol to use when connecting to the remote server:

- None: No security protocol will be used
- SSL, TLS, or PCT: Either the SSL 2.0, SSL 3.0, PCT or TLS protocols will be used when establishing a secure connection. The correct protocol is automatically selected.
- SSL: Either SSL 2.0 or SSL 3.0 may be used when establishing a secure connection. The correct protocol is automatically selected.
- TLS: The TLS 1.0 protocol will be used when establishing a secure connection.
- PCT: The PCT 1.0 protocol will be used when establishing a secure connection.
- SSH: Either SSH 1.0 or SSH 2.0 will be used when establishing a secure connection, based on the version of the protocol that is supported by the server.
- SSH v1: The SSH 1.0 protocol will be used when establishing the connection. This is an older version of the protocol which should not be used unless explicitly required by the server. Most modern SSH server support version 2.0 of the protocol.
- SSH v2: The SSH 2.0 protocol should be used when establishing the connection. This is the default version of the protocol that is supported by most SSH servers.

## Certificate Info

**Location:** The location to retrieve the certificate from (user store, machine store, or PFX file).

**Store name/file:** The name of the store to open or the PFX file to retrieve the certificate from (optional).

**Certificate name:** Friendly name of the certificate to use (optional). Note that the name must match completely, but the comparison is not case sensitive. If no matching certificate is found, the action will then attempt to find a certificate that has a matching common name (also called the certificate subject). This comparison is less stringent, and the first partial match will be returned. If this second search fails, the action will return an error indicating that the certificate could not be found.

**Password:** The certificate password (applies only for certificates in a PFX file).

*Note:* Certificate info does not apply for SSH protocols.

**Verify host fingerprint:** If checked, the specified fingerprint will be compared with the fingerprint value returned by the server, and the step will fail if they do not match. Build the step once to determine the host key, then use the value from the build output here for future use (to prevent man-in-the-middle security attacks). This option applies only for SSH connections.

## 3.12 Server

### 3.12.1 ADO

The ADO action executes database commands or queries using ADO.

When the step completes, the following temporary macros are created or updated:  
ADO\_RS = When performing queries that return results, the ADO recordset object holding the query results

#### Connection tab

#### Command tab

*Notes:*

- See the Server.bld sample for a project utilizing this action.
- This action requires [MDAC](#) 1.5 or later to be installed.

#### 3.12.1.1 Connection Tab

This tab of the ADO action specifies the database connection information.

**Provider:** The OLEDB provider to use (required). Choose a provider from the drop-down list or enter a Connection String parameter value for one of the providers in this provider list).

**Data Source:** The database filename, path, name or URL (required).

**Initial Catalog:** The catalog or database name to connect to (optional).

**Security:** Either SQL Server integrated security can be used (for the SQL Server provider) or an explicit username and password can be specified if needed (if *Use Windows authentication* is checked, Windows authentication will be used rather than database authentication).

**Additional connection properties:** Use this grid to enter any custom connection string properties

names/values (optional) required for your provider. Enter a property name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.1.2 Command Tab

This tab of the ADO action configures the SQL statement to be executed and how to process the results.

**SQL statement:** The SQL statement to execute.

*Note:* Bracket characters [ ] and percent sign % characters within this field must be doubled up (just as in any other field), since these are normally interpreted by Visual Build as referencing script code and macros within a field.

**Statement returns results:** If checked, the results of the query will be stored in an **ADO\_RS** temporary macro. See the Server.bld sample for how to process the results within a build.

**Macro name:** The name of the temporary macro in which to store the ADO recordset object holding the query result (optional, defaults to **ADO\_RS** if not specified).

**Cursor location:** Self-explanatory.

**Cursor type:** Self-explanatory.

**Lock type:** Self-explanatory.

**Command timeout:** Time, in seconds, to wait for a command to execute (optional, default is 30).

## 3.12.2 Amazon

This action creates a step to perform an Amazon Web Services command.

*Note:* The AWS CLI must be installed. This action has been tested with v1 and may work with other versions as well.

### Command Tab

### Options Tab

### Advanced Tab

### Remote Tab

#### 3.12.2.1 Command Tab

This tab of the Amazon action configures the command to perform.

**Command:** The command to execute (required).

**Command parameters:** Any parameters required for the command -- see the AWS CLI documentation.

**Override aws executable filename:** Specifies the full path of the `aws.exe` executable to call if the location of `aws.exe` is not in the PATH environment variable.

### 3.12.2.2 Options Tab

This tab of the Amazon action configures global options.

**Verbose output:** Turn on debug logging.

**Output formatting style:** The formatting style for command output.

**Disable automatic pagination:** Disables automatic pagination.

**Do not verify SSL connection:** By default, the AWS CLI uses SSL when communicating with AWS services. For each SSL connection, the AWS CLI will verify SSL certificates. This option overrides the default behavior of verifying SSL certificates.

**Do not sign requests:** Credentials will not be loaded if this option is checked.

**Override default URL:** Override command's default URL with the given URL.

**Use specific profile:** Use a specific profile from your credential file.

**Additional options:** Specifies additional AWS CLI options.

### 3.12.3 Azure

This action creates a step to perform a Microsoft Azure command.

#### Advanced Tab

#### Remote Tab

**Command:** The command to execute (required).

**Name:** The name of the target for the command (optional).

**Additional options:** Specifies additional Azure CLI command-line options.

**Verbose output:** If checked, logs additional output.

**Json output:** If checked, generates Json output.

**Override Azure executable filename:** Overrides the default executable used (azure).

#### Notes:

- The Azure CLI must be installed. This action has been tested with v0.9 and may work with other versions as well.
- The `azure.cmd` file installed with Azure CLI does not return the exit code of the operation. A modified version of `azure.cmd` is available in the `Tools` directory in the VisBuildPro install path (if *Full installation* is selected during installation). Copy this file to your Azure CLI `wbin` path (typically `C:\Program Files (x86)\Microsoft SDKs\Azure\CLI\wbin`) for proper success/failure detection.

### 3.12.4 COM Register

This action creates a step to register or unregister a COM EXE, DLL, OCX, .NET assembly, type library, or Visual Build user-defined action.

## Advanced Tab

### Remote Tab

**Filename:** The filename of the component, assembly, control, executable, type library, or custom action to register or unregister (*note:* when unregistering a custom user action, this can be the action name or filename).

**Unregister the component or type library:** Determines whether the file or component is registered or unregistered.

**.NET Assembly:** Check this option to register a .NET assembly using RegAsm (otherwise, RegSvr32 will be used to register DLLs).

**Create Codebase entry:** Creates a Codebase entry in the registry (for .NET assemblies only). The Codebase entry specifies the file path for an assembly that is not installed in the global assembly cache. You should not check this option if you will subsequently install the assembly that you are registering into the global assembly cache. The assembly must be a strong-named assembly.

**REG file:** Generates the specified .reg file for the assembly, which contains the needed registry entries (optional, for .NET assemblies only). Specifying this option does not register the assembly.

**TLB file:** Generates a type library from the specified assembly containing definitions of the accessible types defined within the assembly.

**Override executable filename:** Overrides the default executable used for registering DLLs (regsvr32.exe or regasm.exe).

The following logic is performed based on the filename extension:

- For files with an .exe extension, the EXE is called with flags of /regserver or /unregserver.
- For .tlb files, the action loads and registers or unregisters the type library (the Advanced and Remote tabs do not apply).
- For .action components or files with no extension, the user-defined action is registered (by reading the .action file contents and adding to the registry) or unregistered (by removing that action's key from the registry). After registering or unregistering a user action, choose *View | Refresh* on the menu bar to refresh the Actions pane and Step panes with any changes.
- For all other files, the action registers or unregisters the file using `regsvr32.exe` or `regasm.exe` (.NET assemblies).

*Note:* See the VStudio.bld sample for a project utilizing this action.

### 3.12.5 COM+ Application

This action creates a step to manipulate COM+ applications. It supports creating, modifying, deleting, starting, and shutting down applications.

*Notes:*

- See the Server.bld sample for a project utilizing this action.
- This action requires a Windows 2000 (or newer) install with COM+ installed.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### Application tab



**Operation:** The operation to perform, which can be Create/Modify, Delete, Shutdown, or Start (required). Create/modify can be used to create a new application or update the properties of an existing application.

**Computer:** The computer whose Component Services catalog will be updated (optional). Use blank or *My Computer* for the local computer, or select or enter a remote computer name (the list is prepopulated with all computers that have been added to the Component Services console on the local computer).

**Application:** The application name to create or modify (required). The list is prepopulated with all existing applications on the local computer if the Computer field is empty or equal to *My Computer*; click the Refresh button to refresh the list from a remote computer name in the Computer field.

An asterisk (\*) in this field will update all applications on the computer. If \* is specified, delete will attempt to delete all matching applications marked as Deleteable (except the COM+ Explorer application, which can't be deleted but is marked Deleteable); Shutdown and Startup will try to shutdown or start all Server applications.

A regular expression can be entered to match multiple applications.

**Activation type:** The type of application.

**Username:** The identity that the application process will run under (required). Enter *Interactive User* to use the interactive user, or enter a username or domain\username in this field.

**Password:** The password for the user (optional).

**Server process shutdown:** Whether to leave the server running when idle or how long to wait before shutting down an idle server process.

*Note:* The Identity and Server process shutdown option are only meaningful for Server applications.

*Note:* See the Server.bld sample for a project utilizing this action.

## Properties tab

This tab can be used to set additional properties of the application. See the [MSDN help topic on the Applications collection](#) for the list of available properties and values.

## Roles tab

### 3.12.5.1 COM+ Properties Tab

This tab of the COM+ Application and COM+ Component actions is used to add or update custom properties.

Use the drop-down list or see the MSDN help topic for the [Applications collection](#) or the [Components collection](#) for the list of available properties and values. Enter or choose a property name, enter its value, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

Properties can be String, Boolean, or Long values. The action converts values to the correct type in the following manner: True and False are converted to Boolean, an all-numeric value is converted to Long, and all other values are treated as String. To explicitly override this (for instance, to assign an all-numeric string value), prefix the property Value field with S|, B| or I|.

### 3.12.5.2 COM+ Application Roles Tab

This tab of the COM+ Application action is used to add or update roles.

Enter a role name and users for that role (separated by commas or semicolons) and click Insert to add to the list. Select an item in the list and update the user names and click Update to update, or click Delete to remove it from the list.

### 3.12.6 COM+ Component

The COM+ Component action creates a step to create, modify, delete, and report on COM+ components.

*Notes:*

- See the Server.bld sample for a project utilizing this action.
- This action requires a Windows 2000 (or a newer version) install with COM+ installed.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### Component tab

**Operation:** The operation to perform, which can be Add, Modify, Delete, or Report (required).

**Computer:** The computer whose Component Services catalog will be updated (optional). Use blank or *My Computer* for the local computer, or select or enter a remote computer name (the list is prepopulated with all computers that have been added to the Component Services console on the local computer).

**Application:** The application name to modify (required). The list is prepopulated with all existing applications on the local computer if the Computer field is empty or equal to *My Computer*; click the Refresh button to refresh the list from a remote computer name in the Computer field.

Enter an asterisk (\*) to update all matching components in all applications on the computer (cannot be used with the Add operation).

Also, a regular expression can be entered to match multiple applications.

**Component:** The component to add or modify (required). When adding, this must be the full drive, path and filename of the component to be added. For other operations, this can be the DLL filename or a component ProgId. An asterisk (\*) can also be entered to modify, delete, or show all components in the specified application(s). When adding a component, if the type library is in a separate file and/or there is a proxy-stub DLL, enter each filename separated by semicolons in this field: <component filename>;<type library filename>;<proxy-stub filename>.

Or a regular expression can be entered to match multiple components.

**Install as event class:** When adding components, if unchecked, the action installs all components (COM classes) from the DLL file in the Component field into the specified COM+ application and registers the components in the COM+ class registration database as configured components. If checked, the action will install event classes from a DLL holding dummy implementations of the event classes.

**Transaction support:** Transaction support option. If *Default* is chosen, the default or value configured in the component metadata will be used.

**Synchronization support:** Synchronization support option. If *Default* is chosen, the default or value

configured in the component metadata will be used.

**Transaction timeout:** Component-specific transaction timeout value for transactional components (optional). If left blank, the default (computer-level) transaction timeout will be used.

**Enable Just In Time activation:** Whether to enable JIT for the component.

*Note:* Some combinations of properties are not valid (for instance, Transaction support of *Disabled* and Synchronization support of *Supported*), but this screen does not prevent invalid combinations from being selected, and invalid values will be ignored when the properties are assigned. See the link below to determine valid property values or view the component property pages in the Component Services Explorer to see the valid combinations.

*Note:* See the Server.bld sample for a project utilizing this action.

## Properties tab

This tab can be used to set additional properties of the component. See the [MSDN help topic on the Components collection](#) for the list of available properties and values.

## Roles tab

### 3.12.6.1 COM+ Component Roles Tab

This tab of the COM+ Component action is used to add or update roles.

Enter a role name and click Insert to add to the list. Select an item in the list and click Delete to remove it from the list.

### 3.12.7 IIS Virtual Dir

The IIS Virtual Dir action unloads, deletes, creates, or updates an IIS virtual directory.

## IIS Tab

## Options tab

*Notes:*

- This action requires Microsoft [IIS](#) v4 or later to be installed.
- For IIS v7 and later, use the IIS action, or Metabase Compatibility Support must be installed: In *Control Panel -> Programs -> Programs and Features*, click *Turn Windows features on or off*, check the option *Internet Information Services -> Web Management Tools -> IIS 6 Management Console -> IIS Metabase and IIS configuration compatibility* and click OK.
- See the Server.bld and VStudio.bld samples for a project utilizing this action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.12.7.1 IIS Tab

This tab of the IIS Virtual Dir Action action specifies the server and directory settings.

**Server:** The name of a remote server to connect to (optional).

**Web site #:** The web site number when not using the Default web site.

**Operation:** The operation to perform (unload, delete, or create/update).

**Directory name:** The name of the virtual directory (optional, updates default web site if blank).

**Local path:** The local path that the virtual directory will point to (required when creating or updating).

**Application protection:** The application protection level for the virtual directory.

### 3.12.7.2 Options Tab

This tab of the IIS Virtual Dir Action action configures additional directory settings.

**Read access:** Whether read access will be available for the directory.

**Write access:** Whether write access will be available for the directory.

**Execute access:** Whether execution of scripts and executables will be allowed for the directory.

**Script source access:** Whether script source code access will be available for the directory.

### 3.12.8 IIS

The IIS action performs a command on a Microsoft Internet Information Server web server.

#### IIS Tab

#### Options Tab

#### Properties Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- This action requires Microsoft [IIS](#) v7 or later to be installed.
- Use the IIS Virtual Dir action for creating virtual directories in older versions of IIS.
- Many commands require administrator (elevated) privileges.

#### 3.12.8.1 IIS Tab

This tab of the IIS Action action specifies the command to perform.

**Object:** The object to perform the command on.

**Command:** The command to perform.

**Name/ID:** The ID or name of the object to operate on (required for most commands).

**Path:** The path of the object (required for some commands).

**Location:** Specifies the level that configuration settings are written at. The configuration system is hierarchical, allowing configuration settings to be written at multiple levels ranging from the server-level applicationHost.config file to distributed web.config files that can be present at the site, application, or virtual directory levels. When the configuration is written at a particular level, it is inherited by all URLs

at that level and lower. For example, configuration set in the web.config configuration file at the root of the site is inherited by all URLs of the site. By default, settings will be stored on the level at which it is being set. For example, if you are setting configuration for the "Default Web Site/", it will be written in a web.config file at the root of that site. However, it is possible to write configuration at a higher level, and only apply it to a particular subset of URLs below by using a location construct. For example, the application web.config can contain configuration that is applied to only a single directory within that application. AppCmd provides this capability through its commit parameter.

The Location can be set to one of the following:

- (Blank): write configuration at the level for which it is set
- Site: write configuration in the web.config at the site root of the URL for which it is set
- Application: write configuration in the web.config at the app root of the URL for which it is set
- Server: write configuration at the server level, in the applicationHost.config file
- <PATH>: write configuration at the specified configuration path

### 3.12.8.2 Options Tab

This tab of the IIS Action action configures additional options.

**Override default IIS command-line executable:** If left blank, the action will automatically locate the IIS command-line executable. By providing a full path and filename for this field, that executable will be used instead.

**Additional command-line options:** Used to specify additional flags to pass to `Appcmd.exe` (optional).

### 3.12.8.3 Properties Tab

This tab of the IIS Action action specifies properties to set or filter on.

**Property names/values:** Use this grid to enter any custom property names/values to filter or set (optional). Enter a property name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

## 3.12.9 Run Oracle Script

The Run Oracle Script action creates a step to run SQL scripts against an Oracle database. It generates a SQL\*Plus statement from the inputs and runs it.

### Input Tab

### Options Tab

### Advanced Tab

### Remote Tab

This action has been tested with Oracle v10 and v11 and may work with other versions as well.

#### 3.12.9.1 Input Tab

This tab of the Run Oracle Script action configures the database and script to be executed.

**User ID:** The user account to login as (required).

**Password:** The user's password (optional).

**Database/SID:** The database or SID to connect to (optional).

**Script file:** The SQL script file to run (required).

**Parameters:** Parameter values to pass to the SQL script (optional).

### 3.12.9.2 Options Tab

This tab of the Run Oracle Script action configures additional options.

**Additional command-line options:** Allows for any additional SQL\*Plus command options to be entered and passed through. See the [SQL\\*Plus documentation](#) for details on the available options.

**SQL\*Plus executable filename:** If not specified, the action will invoke the `sqlplus.exe` command-line tool (it must be in the PATH environment variable). This can be overridden by providing the executable filename and path in this field.

## 3.12.10 Run SQL

The Run SQL action creates a step to run SQL scripts against a Microsoft [SQL Server](#) database. It generates a SQLCMD statement from the inputs and runs it. The SQL script can be provided within the step (using the built-in editor with SQL syntax highlighting) or by referencing a separate file containing the script.

### Server tab

### Input tab

### Output tab

### Options tab

### Advanced Tab

### Remote Tab

This action requires the SQL Server client tools to be installed. It has been tested with SQL Server 2000 thru 2014 and may also work with other versions.

#### Notes:

- Use the ADO action for querying a SQL Server database or modifying or querying other databases from a build.
- See the Server.bld sample for a project utilizing this action.

### 3.12.10.1 Server Tab

This tab of the Run SQL action specifies server and login settings.

**Server:** The database server to connect to (optional). If not specified, the SQL Server checks for the environment variables and uses those, for example, `osqluser=(user)` or `osqlserver=(server)`. If no environment variables are set, the workstation user name is used.

**Database:** The name of the database to access (optional). If you do not specify a server, the name of the workstation is used.

**Security:** Either SQL Server integrated security can be used or an explicit username and password can be specified.

**Abort batch on error:** Specifies that SQLCMD exits with a failure exit code when an error occurs. The exit code returned is 1 when the SQL Server error message has a severity of 10 or greater; otherwise, the value returned is 0.

**Show numbering:** If unchecked, removes numbering and the prompt symbol (>) from input lines.

### 3.12.10.2 Input Tab

This tab of the Run SQL action configures the SQL statements to be executed.

This tab is used to specify the SQL statements to be executed on the server. This can be a SQL string or a filename containing the SQL statements (required). A Transact SQL reference is available as well as common SET commands that can be used.

*Note:* Bracket characters [ ] and percent sign % characters within this field must be doubled up (just as in any other field), since these are normally interpreted by Visual Build as referencing script code and macros within a field.

### 3.12.10.3 Output Tab

Use this tab to redirect the output from the command to a file if desired.

**Send output to:** Standard output or a file. With either choice, the output will be logged by Visual Build and will be available in the LASTSTEP\_OUTPUT system macro.

**Column width:** Specifies the screen width for output (optional). The column width must be a number greater than 8 and less than 65536. If the specified column width does not fall into that range, sqlcmd generates an error message. The default width is 80 characters. When an output line exceeds the specified column width, it wraps on to the next line.

**Column separator:** Specifies the column-separator character (optional). The default is a blank space. To use characters that have special meaning to the operating system such as the ampersand (&), or semicolon (;), enclose the character in quotation marks ("). The column separator can be any 8-bit character.

**Fixed-length type display width:** Specifies the display width for fixed length fields (optional). The default is 0 (unlimited). Limits the number of characters that are returned for the following data types:

- char(n), where  $1 \leq n \leq 8000$
- nchar(nn), where  $1 \leq n \leq 4000$
- varchar(nn), where  $1 \leq n \leq 8000$
- nvarchar(nn), where  $1 \leq n \leq 4000$
- varbinary(nn), where  $1 \leq n \leq 4000$
- variant

**Variable-length type display width:** Specifies the display width for variable length fields (optional). The default is 256. A value of 0 will truncate the output at 1MB. It limits the number of characters that are returned for the large variable length data types:

- varchar(max)
- nvarchar(max)
- varbinary(max)
- xml
- UDT (user-defined data types)

- text
- ntext
- image

**No column headers:** If checked, no column headers will be printed.

**Print performance statistics:** If checked, prints performance statistics.

#### 3.12.10.4 Options Tab

This tab of the Run SQL action specifies additional options.

**Override the path/filename of the SQLCMD command-line executable:** If not specified, the action will invoke the `SQLCMD.EXE` command-line tool (`sqlcmd.exe` must be in the PATH environment variable). This can be overridden by providing the executable filename (and path if desired) in this field.

**Additional command options:** Allows for any additional command options to be entered and passed through. See the SQLCMD documentation or [OSQL documentation](#) for details on the available options.

#### 3.12.11 Service

This action creates a step to manage a Windows service on the local computer or another computer on the network. Select or enter the computer name (or enter blank for the Computer for the local computer) and service name and choose the operation to perform. The service name can be the display name as shown in the *Name* column of the *Services* management console or the short service name (when creating a service, enter the short name in this field).

When the step completes, the following temporary macros are created or updated for the *Query status* operation:

**SERVICE\_STATUS:** Holds the service status (1 = stopped, 2 = start pending, 3 = stop pending, 4 = running, 5 = continue pending, 6 = pause pending, 7 = paused).

**SERVICE\_START\_TYPE:** Holds the service startup type (0 = boot, 1 = System start, 2 = Automatic, 3 = Manual, 4 = Disabled).

##### Service tab

##### Properties tab

*Notes:*

- See the `Server.bld` sample for a project utilizing this action.
- To access services on a remote computer, the Remote Procedure Call service on the remote computer must be started. Your network configuration may also need to be adjusted to allow access through Windows Firewall.

#### 3.12.11.1 Service Tab

This tab of the Service action specifies information about the service and operation.

**Computer:** The name of the computer to start or stop a service on (optional; blank for local computer).

**Service:** The display name of the service to start or stop (required). Click *Refresh* to populate the drop-down list of services for the selected computer.

**Operation:** The operation to perform on the service (Start, Stop, Pause, Resume, Query status, Create, Configure, Delete). For the *Query status* operation, the service status (1 = stopped, 2 = start pending, 3 = stop pending, 4 = running, 5 = continue pending, 6 = pause pending, 7 = paused) is



stored in the SERVICE\_STATUS temporary macro, and the service startup type (0 = boot, 1 = System start, 2 = Automatic, 3 = Manual, 4 = Disabled) is stored in the SERVICE\_START\_TYPE temporary macro.

**Timeout:** The maximum amount of time to wait for the service to perform the operation (optional; defaults to 100 retries on wait hint provided by service if blank or 0).

**Username:** Specifies the user account that will be used to connect to the service database (optional). If blank, the current user identity will be used. To specify a domain, use the format domain@username.

**Password:** Specifies the password of the user account (optional).

### 3.12.11.2 Properties Tab

This tab of the Service action sets properties when creating or configuring a service.

**Service Type:** Type of service to create.

**Runs in own process:** Indicates that the service runs in its own process.

**Shares a process with other service:** Indicates that the service shares a process with other services.

**Startup type:** Service start options.

**Error control:** Severity of the error, and action taken, if this service fails to start

**Executable:** The fully qualified path and filename of the service executable (required when creating a service). If the path contains a space, it must be quoted so that it is correctly interpreted. This field can also include arguments for an auto-start service, which are passed to the service entry point (typically the main function).

**Display name:** The description of the service (optional; used to identify the service when starting, stopping, etc.).

**Account:** The name of the account under which the service should run (optional; uses LocalSystem account if blank). If the *Runs in own process* is checked, use an account name in the form DomainNameUserName. The service process will be logged on as this user. If the account belongs to the built-in domain, you can specify .\UserName. If the service type is Kernel driver or File System driver, the name is the driver object name that the system uses to load the device driver (or blank if the driver is to use a default object name created by the I/O system).

**Allow service to interact with desktop:** If either *Runs in own process* or *Shares a process with other services* is checked, this option is enabled and specifies that the service can interact with the desktop.

**Password:** The password to the account name specified by the Account field (optional). Specify an empty string if the account has no password. Passwords are ignored for driver services.

**Group:** Names the load ordering group of which this service is a member (optional). Specify an empty string if the service does not belong to a group.

The startup program uses load ordering groups to load groups of services in a specified order with respect to the other groups. The list of load ordering groups is contained in the ServiceGroupOrder value of the following registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control
```

**Dependencies:** Semicolon-separated list of names of services that the system must start before this service (optional).

### 3.12.12 Web Deploy

The Web Deploy action invokes the Microsoft Web Deploy tool, which simplifies the migration, management and deployment of IIS web servers, web applications and web sites.

#### Source Tab

#### Destination Tab

#### Parameters Tab

#### Link Extensions Tab

#### Rules Tab

#### Skip Tab

#### Commands Tab

#### Miscellaneous Tab

#### Options Tab

*Note:* This action has been tested with Web Deploy v1 through v3 and may work with other versions as well.

#### 3.12.12.1 Source Tab

This tab of the Web Deploy action specifies the verb and source information.

**Verb:** The verb to perform (required).

**Source object provider:** The source object provider (required).

**Source provider value:** A value for the source provider (optional).

**Source provider settings and values:** Use this grid to enter any custom provider names/values (optional). Enter a name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.12.12.2 Destination Tab

This tab of the Web Deploy action configures destination options.

**Destination object provider:** The destination object provider (required for sync command).

**Destination provider value:** A value for the destination provider (optional).

**Destination provider settings and values:** Use this grid to enter any custom provider names/values (optional). Enter a name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.12.3 Parameters Tab

This tab of the Web Deploy action specifies additional parameters.

**Declare parameter names/values:** Use this grid to enter any declare parameter names/values (optional). Enter a parameter name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Set parameter names/values:** Use this grid to enter any custom set parameter names/values (optional). Enter a parameter name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Remove parameter names:** Use this grid to enter any parameters to remove (optional). Enter a parameter name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.12.4 Link Extensions Tab

This tab of the Web Deploy action specifies link extensions to enable or disable.

**Disable link extensions:** Use this grid to enter any extensions to disable (optional). Enter an extension name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Enable link extensions:** Use this grid to enter any extensions to enable (optional). Enter an extension name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.12.5 Rules Tab

This tab of the Web Deploy action configures rules to enable, disable, or remove.

**Disable rules:** Use this grid to enter any rules to disable (optional). Enter a rule name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Enable rules:** Use this grid to enter any rules to enable (optional). Enter a rule name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.12.6 Skip Tab

This tab of the Web Deploy action specifies skip directives.

**Use skip directive:** Use this grid to enter any custom skip directives to use (optional). Enter a name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Enable skip directive:** Use this grid to enter any skip directives to disable (optional). Enter a name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Disable skip directive:** Use this grid to enter any skip directive to disable (optional). Enter a name in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.12.7 Commands Tab

This tab of the Web Deploy action specifies commands to execute before or after synchronization.

**Command to execute before synchronization:** Specifies a command to execute before

synchronization (optional).

**Command to execute after synchronization:** A command to execute after synchronization (optional).

### 3.12.12.8 Miscellaneous Tab

This tab of the Web Deploy action configures miscellaneous options.

**Retry attempts:** The number of retry attempts, in milliseconds (optional, defaults to 1000).

**Retry interval:** How long to wait between retries, in milliseconds (optional, defaults to 1000).

**Replace attributes:** Use this grid to enter any attributes to replace (optional). Enter a name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.12.12.9 Options Tab

This tab of the Web Deploy action specifies additional options.

**Log what would happen but don't perform operation:** Displays the projected results of an operation, but does not perform the operation.

**Verbose output:** Outputs all available information about the operation (info, warnings, and errors).

**Allow untrusted server certificate when using SSL:** Self-explanatory.

**Return results in XML format:** Self-explanatory.

**Show secure attributes in XML:** Causes secure attributes to be displayed in clear text instead of asterisks during XML output.

**XPath expression to apply to the XML output:** Self-explanatory.

**Property names/values:** Use this grid to enter any custom property names/values to filter or set (optional). Enter a property name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

**Additional command-line options:** Used to specify additional flags to pass to `MSDeploy.exe` (optional).

**MS Deploy executable filename:** If left blank, the action will automatically locate the MS Deploy command-line executable. By providing a full path and filename for this field, that executable will be used instead.

*Note:* To call the 64-bit version of `MSDeploy.exe` from the 32-bit edition of Visual Build, enter the full path and filename of the 64-bit version of `MSDeploy.exe` (i.e., `C:\Program Files\IIS\Microsoft Web Deploy\msdeploy.exe`). To call the 32-bit version of `MSDeploy.exe` from the 64-bit edition of Visual Build, enter the full path and filename of the 32-bit version of `MSDeploy.exe` (i.e., `C:\Program Files (x86)\IIS\Microsoft Web Deploy\msdeploy.exe`).

## 3.13 Test Driven Development

### 3.13.1 Ant

This action executes [Ant](#) builds. The action provides ways to configure the various options for Ant. Before using this action:

1. Download and extract the Ant binaries.
2. Add the Ant `bin` directory to your `PATH` environment variable.
3. Create an `ANT_HOME` environment variable pointing to the Ant installation path.

#### Input Tab

#### Output Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

##### Notes:

- The `ant.bat` file installed with Ant does not return the exit code of the build result, but rather the exit code of later commands in the batch file. A modified version of `ant.bat` is available in the `Tools` directory in the VisBuildPro install path (if *Full installation* is selected during installation). Copy this file to your Ant `bin` path for proper success/failure detection of Ant builds.
- This action has been tested with Ant versions 1.6 thru 1.9 and may work with other versions as well.
- See the `TDD.bld` sample for a project utilizing this action.

#### 3.13.1.1 Input Tab

This tab of the Ant action configures input settings.

**Build file:** Use the specified build file (required).

**Find:** If checked, searches for the build file towards the root of the file system.

**Targets:** Targets to build (optional, one per line if specified).

**Property names/values:** Assigns values to properties (optional). Java system properties are listed here.

#### 3.13.1.2 Output Tab

This tab of the Ant action configures output settings.

**Execute all targets not depending on failed targets:** Self-explanatory.

**Library paths:** Specifies paths to search for jars and classes (optional, one per line).

**Logger component:** The class which is to perform logging (optional).

**Log file:** File to log to (optional).

**Quiet mode:** Be extra quiet.

**Verbose mode:** Be extra verbose.

**Emacs output mode:** Produce logging information without adornments.

**Debug mode:** Print debugging information.

### 3.13.1.3 Options Tab

This tab of the Ant action configures additional options.

**Listeners:** Classes to add as project listeners (optional, one per line).

**Input handler:** The class that will handle input requests (optional).

**Don't use jar files from `${user.home}/.ant/lib`:** Self-explanatory.

**Run without using CLASSPATH:** Self-explanatory.

**Output diagnostic information:** Log information that might be helpful for diagnosing or reporting problems.

**Output a list of targets:** Log project help information.

**Additional options:** Used to specify any additional Ant command-line options.

**Override default Ant.bat location:** Identifies the Ant.bat file to invoke (optional). If not provided, the location of Ant.bat must be found in the PATH environment variable.

## 3.13.2 FxCop

This action performs FxCop code analysis. The command-line application (`FxCopCmd.exe`) is invoked. The action is used to configure the options available for the console version of FxCop. Download and install FxCop before using.

### Input Tab

### Output Tab

### Options Tab

### Advanced Tab

### Remote Tab

*Notes:*

- This action has been tested with FxCop versions 1.32 and 1.35 and may work with other versions as well.
- See the TDD.bld sample for a project utilizing this action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.13.2.1 Input Tab

This tab of the FxCop action configures input settings.

**Project file:** Specifies a `.fxcop` project file to use (optional unless the Target and Rule fields are empty).

**Assemblies or folders:** Specifies a list of assemblies or folders to analyze (optional unless the project file is not specified, one per line). For folders, FxCop attempts to analyze all files with `.exe` and `.dll` file extensions.

**Rule library files or folders:** Specifies locations of rule libraries to load (optional unless project file is not specified, one per line). For folders, all files with a `.dll` extension in the folder are loaded.

**Additional folders to search for assembly dependencies:** Specifies additional folders to search for assembly dependencies (in addition to the target assembly folder and the current working directory; optional, one per line).

### 3.13.2.2 Output Tab

This tab of the FxCop action configures output settings.

**Types to analyze:** Specifies the types to analyze (optional). This option disables analysis of assemblies, namespaces, and resources; only the specified types and their members are included in the analysis. The list can use the wildcard character `*` at the end of the name to select multiple types.

**Analysis report filename:** Specifies the filename for the analysis report (optional). If the file exists, it is overwritten without warning. If no items are reported by the analysis and the file does not exist, it is not created. If the file exists, it is deleted. By default, the file includes an xml-stylesheet processing instruction that references `FxCopReport.xml`.

**XSL file referenced by analysis report:** Specifies the XSL or XSLT file that is referenced by the xml-stylesheet processing instruction in the analysis report (optional). This option overrides the default XSL file applied to the report. Choose `none` to generate a report without a style sheet processing instruction.

**Apply XSL transformation before saving:** Applies the specified XSL transformation to the analysis report before saving the file.

**Include summary report:** Includes a summary report with the informational messages. The summary shows the number of items found, how many items were new, and the running time for the analysis.

**Verbose output:** Outputs verbose information during analysis.

**Send analysis output to Output pane:** Directs analysis output to the Output pane. By default, the XSL file `FxCopConsoleOutput.xml` is applied to the output before it is logged.

**XSL file to apply to analysis output:** Specifies the XSL or XSLT file that contains a transformation to be applied to the analysis output before it is displayed in the Output pane (optional). This option overrides the default XSL file applied to the analysis output.

### 3.13.2.3 Options Tab

This tab of the FxCop action configures additional options.

**Save results in project file:** Saves the results of the analysis in the project file. This option is ignored if the a project file is not specified.

**Analysis reports, projects, or folders to import:** Specifies the names of analysis reports, project files, or folders to import (optional, one per line). Any messages in the imported file that are marked as excluded are not included in the analysis results. If you specify a folder, FxCop attempts to import all files with the `.xml` extension. To import all FxCop project files instead, include `*.FxCop`. If analysis

results are saved to the project file, the imported messages are not saved.

**Location of Mscorlib.dll:** Location of the version of `Mscorlib.dll` that was used when building the target assemblies if this version is not installed on the computer running FxCop (optional).

**Additional options:** Used to specify any additional FxCop command-line options.

**FxCop executable:** Overrides the default location of `FxCopCmd.exe` (optional). Useful if multiple versions of FxCop are installed.

### 3.13.3 Gallio

This action executes Gallio unit tests, which can run tests from MbUnit, MSTest, NBehave, NUnit, xUnit.Net, csUnit, and RSpec. The action provides ways to fully configure the various options available for the console version of Gallio. Download and install Gallio before using.

#### Input Tab

#### Runner Tab

#### Reports Tab

#### Misc Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- This action has been tested with Gallio version 3 and may work with other versions as well.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.13.3.1 Input Tab

This tab of the Gallio action configures input settings.

**Assemblies and test files to run:** Specifies one or more assemblies or test files to run (required).

**Working directory:** The working directory to use during test execution instead of the default (optional).

**Plugin directories:** Additional plugin directories to search recursively (optional).

#### 3.13.3.2 Runner Tab

This tab of the Gallio action configures runner settings.

**Test runner type:** Specifies the type of test runner to use (optional; the default is *IsolatedProcess*).

**Test runner extensions:** Specifies the type, assembly, and parameters of custom test runner extensions to use during the test run (optional).



**Time limit:** Maximum amount of time (in seconds) the tests can run before they are canceled (optional; the default is an infinite time to run).

**Test runner properties and values:** Specifies options property keys/values for the test runner (optional). Enter a property key and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.13.3.3 Reports Tab

This tab of the Gallio action configures report settings.

**Report types:** Report types to generate (optional).

**Report directory:** Target output directory for the reports (optional; default is *Reports*).

**Report name format:** Format string for the report name (optional; {0} is replaced by the date, {1} by the time; default is *test-report-{0}-{1}*).

**Report formatter properties:** Specifies property keys/values for the report formatters (optional). Enter a property key and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.13.3.4 Misc Tab

This tab of the Gallio action configures miscellaneous options.

**Show generated reports:** Show generated reports in a window using the default system application registered to the report file type.

**Do not echo results:** Do not echo results as tests finish. If this option is specified, only the final summary statistics are displayed. Otherwise test results are echoed to the console in varying detail depending on the current verbosity level.

**Do not display progress:** Do not display progress messages during execution.

**Filter set to apply:** Sets the filter set to apply, which consists of one or more inclusion or exclusion filter rules prefixed using *include* (optional) or *exclude*. A filter rule consists of zero or more filter expressions that may be combined using *and*, *or*, and *not* and grouped with parentheses.

**Hint directory:** Additional directories used for loading referenced assemblies and other dependent resources (optional).

### 3.13.3.5 Options Tab

This tab of the Gallio action configures additional options.

**.Net runtime version to run under:** Specifies the version of the .Net runtime to use for running tests (optional). For the CLR, this must be the name of one of the framework directories in `%SystemRoot%\Microsoft .Net\Framework` (e.g., *v2.0.50727*).

**Verbosity level:** Controls the level of detail of the information to display.

**Do not run tests:** Load the tests but does not run them. This option may be used to produce a report that contains test metadata for consumption by other tools.

**Ignore annotations when determining result code:** Ignore annotations when determining the result code. When not specified error annotations, usually indicative of broken tests, will cause a failure result to be generated.

**Enable shadow copying of assemblies:** Enable shadow copying of the assemblies. Shadow copying allows the original assemblies to be modified while the tests are running. However, shadow copying may occasionally cause some tests to fail if they depend on their original location.

**Attach debugger to the test process:** Attach the debugger to the test process.

**Additional options:** Used to specify any additional Gallio command-line options (optional).

**Gallio executable:** Overrides the default location of `Gallio.Echo.exe` (optional). Useful if multiple versions are installed.

### 3.13.4 Gradle

This action executes Gradle builds. The action provides ways to configure the various options for Gradle. Before using this action, download Gradle.

#### Gradle Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Note:* This action has been tested with Gradle versions 0.9 thru 2 and may work with other versions as well.

#### 3.13.4.1 Gradle Tab

This tab of the Gradle action configures goals and related settings.

**Working directory:** The folder to execute Gradle in and look for the build.gradle file (optional).

**Tasks:** Tasks to perform (required, one per line). Examples: tasks, properties, dependencies, projects, etc.

Dry run:

Do not rebuild project dependencies:

Don't search in parent folders for settings file:

**Log level:**

#### 3.13.4.2 Options Tab

This tab of the Gradle action configures additional options.

**Ignore task optimization:** .

**Settings file:** Alternate user settings file (optional).

**JVM system property names/values:** Assigns values to JVM system properties (optional).

**Additional options:** Used to specify any additional Gradle command-line options.

**Override default gradle.bat location:** Identifies the gradle.bat file to invoke (optional). If not provided, the location of gradle.bat must be found in the PATH environment variable.

### 3.13.5 Maven

This action executes Maven builds. The action provides ways to configure the various options for Maven. Before using this action, download, install, and configure Maven.

#### Maven Tab

#### Projects Tab

#### Flags Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Note:* This action has been tested with Maven versions 2 and 3 and may work with other versions as well.

#### 3.13.5.1 Maven Tab

This tab of the Maven action configures goals and related settings.

**Working directory:** The folder to execute Maven in (optional).

**Goals:** Goals or lifecycle phases to execute (required, one per line). Examples: clean, validate, compile, test, verify, install, deploy, site.

**Profiles to activate:** Profiles to activate (optional, one per line).

**Property names/values:** Assigns values to Maven properties (optional).

#### 3.13.5.2 Projects Tab

This tab of the Maven action configures project settings.

**Reactor projects to build:** Build specified reactor projects instead of all projects (optional, one per line).

**Do not recurse into sub-projects:** Self-explanatory.

**Also build required projects:** Also build projects required by the list.

**Also build dependent projects:** Also build projects that depend on projects in the list.

**Dynamically build reactor from subdirectories:** Self-explanatory.

**Resume reactor from project:** Self-explanatory (optional).

### 3.13.5.3 Flags Tab

This tab of the Maven action configures additional flags.

**Failure handling:** Controls how Maven reacts to a build failure in the middle of a multi-module project build.

**If checksums don't match:** Affects the way Maven will interact with remote repositories and how it verifies downloaded artifacts.

**Plugin up-to-date check:** Tells Maven how it should update (or not update) Maven plugins from remote repositories.

**Logging level:** Controls the Maven logging level.

**Force a check for updated releases and snapshots on remote repositories:** Self-explanatory.

**Offline mode:** Work offline.

**Master security password:** Self-explanatory.

**Server password:** Self-explanatory.

### 3.13.5.4 Options Tab

This tab of the Maven action configures additional options.

**POM file:** Force the use of an alternative POM file (optional).

**User settings file:** Alternate user settings file (optional).

**Global settings file:** Alternate global settings file (optional).

**Additional options:** Used to specify any additional Maven command-line options.

**Override default mvn.bat location:** Identifies the mvn.bat file to invoke (optional). If not provided, the location of mvn.bat must be found in the PATH environment variable.

## 3.13.6 NAnt

This action executes [NAnt](#) builds. The action provides ways to fully configure the various options for NAnt. Before building:

- 1) Download and extract the NAnt binaries.
- 2) Create a `NANT_PATH` environment variable or project/global macro pointing to `<NAnt dir>\bin`.

### Input Tab

### Output Tab

### Options Tab

### Advanced Tab

### Remote Tab

*Notes:*

- This action has been tested with NAnt versions 0.85 through 0.92 and may work with other versions as well.
- See the TDD.bld sample for a project utilizing this action.

### 3.13.6.1 Input Tab

This tab of the NAnt action configures input settings.

**Build file:** The build file to build (required).

**Find:** If checked, searches for the build file towards the root of the file system.

**Targets:** Targets to build (optional, one per line if specified).

**Property names/values:** Assigns values to properties (optional).

### 3.13.6.2 Output Tab

This tab of the NAnt action configures output settings.

**Framework to target:** Specify .NET Framework version to target (optional).

**Logger component:** The logger component to use (optional).

**Log file:** The log file to write output to (optional).

**Indent level:** Indent level for build output (optional).

**Quiet mode:** Be extra quiet.

**Verbose mode:** Be extra verbose.

**Emacs output mode:** Produce logging information without adornments.

**Debug mode:** Print debugging information.

**Show logo:** Suppresses display of logo banner if unchecked.

### 3.13.6.3 Options Tab

This tab of the NAnt action configures additional options.

**Listeners:** Add specified classes as project listeners (optional, one per line).

**Extensions:** Load NAnt extensions from the specified assemblies (optional, one per line).

**Additional options:** Used to specify any additional NAnt command-line options.

**NAnt executable:** Overrides the default location of `NAnt.exe` (optional). Useful if multiple versions of NAnt are installed.

## 3.13.7 NCover

This action performs [NCover](#) code coverage analysis. The command-line application (`NCover.Console.exe`) is invoked. The action allows configuring the various options available for the console version of NCover. Download and install NCover before using.

## Input Tab

## Output Tab

## Options Tab

## Advanced Tab

## Remote Tab

### Notes:

- This action has been tested with NCover versions 1 through 3 and may work with other versions as well.
- See the TDD.bld sample for a project utilizing this action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.13.7.1 Input Tab

This tab of the NCover action configures input settings.

**Mode:** Specifies the mode to operate in.

**Executable + parameters:** Used to specify the executable and parameters for Executable mode or service name for Service mode.

**Working directory:** Specifies the working directory for the profile application.

**List of assemblies to profile:** Specifies a list of assemblies to profile (optional, one per line).

**Classes and methods to exclude from coverage:** Specifies a list of attributes marking classes or methods to exclude from coverage (optional, one per line).

### 3.13.7.2 Output Tab

This tab of the NCover action configures output settings.

**Logging level:** Self-explanatory.

**Profile log file:** Specifies a log file to write output to (optional).

**Coverage output file:** Specifies the output file to write coverage information to (optional).

**HTML report output path:** Specifies a folder to output an HTML report to (optional).

### 3.13.7.3 Options Tab

This tab of the NCover action configures additional options.

**Settings file:** Specifies settings file to use (optional).

**Save settings to file:** Saves configured settings to the specified file (optional).

**Additional options:** Used to specify any additional NCover command-line options.

**NCover executable:** Overrides the default location of `NCover.exe` (optional). Useful if multiple versions are installed.

### 3.13.8 NDepend

This action performs NDepend code analysis. Download and extract NDepend before using.

#### Input Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Note:* This action has been tested with NDepend versions 2 thru 6 and may work with other versions as well.

#### 3.13.8.1 Project Tab

This tab of the NDepend action configures project settings.

**Project filename:** Specifies the NDepend project file to process (required).

**Override project input directories:** Specifies input directories to use (optional, overrides project settings).

**Override project output directory:** Specifies the output directory (optional, overrides project settings).

**Emit application dependencies graph XML file:** Generates an XML file containing the dependencies graph of the application (optional).

**View HTML report:** Displays HTML report if checked.

#### 3.13.8.2 Options Tab

This tab of the NDepend action configures additional options.

**Quiet mode:** Disable console output.

**Custom XSL for building report:** Specifies a custom XSL file to build report (optional).

**Additional options:** Used to specify any additional NDepend command-line options (optional).

**NDepend executable:** Overrides the default location of `NDepend.Console.exe` (optional). Useful if multiple versions are installed or if the executable is not located in the `PATH` environment variable.

### 3.13.9 NDoc

This action generates [NDoc](#) documentation. It provides ways to configure the various options available. Download and install NDoc for Windows via the `.msi` file before using.

#### Input Tab

## Documentor Tab

## Options Tab

## Advanced Tab

## Remote Tab

### Notes:

- This action has been tested with NDoc version 1.3 and may work with other versions as well.
- See the TDD.bld sample for a project utilizing this action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.13.9.1 Input Tab

This tab of the NDoc action configures input settings.

**Project or folder:** Specifies the NDoc project or a folder containing assemblies to document (optional unless Assemblies not specified).

**Assemblies:** Specifies a list of assemblies to document (optional unless Project/Folder not specified, one per line). Followed by a comma and the name of the compiler-generated doc file (if not named with same root as assembly and .xml extension).

**Max folder recurse depth:** Specifies the maximum folder depth to recurse into (0 = infinite).

**Reference folders:** Specifies a list of folders where referenced assemblies can be located (optional, one per line).

### 3.13.9.2 Documentor Tab

This tab of the NDoc action configures documentor settings.

**Namespace summaries file:** Specifies the namespace summary XML document (optional).

**Documentor:** Specifies the name of the documentor to use (optional).

**Verbose:** Causes full progress information to be displayed during the build.

**Property names/values:** Specifies property names and values to set on documentor (optional, case-sensitive).

### 3.13.9.3 Options Tab

This tab of the NDoc action configures additional options.

**Additional options:** Used to specify any additional NDoc command-line options.

**NDoc executable:** Overrides the default location of `NDocConsole.exe` (optional). Useful if multiple versions are installed.



### 3.13.10 NUnit

This action executes [NUnit](#) unit tests. The command-line application (`nunit-console.exe`) is invoked. The action provides ways to fully configure the various options available for the console version of NUnit. Download and install NUnit (for Windows via the `.msi` file) before using.

#### Input Tab

#### Include/Exclude Tab

#### Output Tab

#### Options Tab

#### Options (More) Tab

#### Advanced Tab

#### Remote Tab

##### Notes:

- This action has been tested with NUnit version 2.2 through 3 and may work with other versions as well.
- The `clr.bat` file installed with NUnit 2.4.8 and earlier does not return the exit code of the test result, but rather the exit code of later commands in the batch file. A modified version of `clr.bat` is available in the `Tools` directory in the VisBuildPro install path (if *Full installation* is selected during installation). Copy this file to your NUnit `bin` path for proper success/failure detection of NUnit tests.
- See the `TDD.bld` sample for a project utilizing this action.

#### 3.13.10.1 Input Tab

This tab of the NUnit action configures input settings.

**Assemblies or projects to run:** Specifies one or more assemblies, projects, or test projects to run (required).

**Tests and assemblies to run:** Instead of specifying assemblies or projects, you may specify a test to be run by providing the name of the test and its containing assembly. The name of the test to be run may be that of a test case, test fixture or a namespace. Enter or choose a test and assembly name and click Insert to add to the list. Select a test in the list to update its assembly or delete it from the list.

**Fixture:** Name of the test fixture and containing assembly to load (optional).

**Configuration:** Specifies the project configuration to use (optional, uses the first configuration found if not specified).

#### 3.13.10.2 Include/Exclude

This tab of the NUnit action configures categories to include and/or exclude.

**Include categories:** Includes tests belonging to the specified categories (optional, one per line).

**Exclude categories:** Excludes tests belonging to the specified categories (optional, one per line).

**Category list type:** Specifies whether the next field list categories to include or exclude (for

compatibility only; use the above Include and Exclude fields for new steps).

**Include/exclude categories:** Specifies categories to include or exclude, depending on the *Category list type* field (optional, one per line).

### 3.13.10.3 Output Tab

This tab of the NUnit action configures output settings.

**Write test case names to output:** Logs an identifying label at the start of each test case.

**XML output file:** Specifies the filename to log output to (optional). If not specified, the file is named `TestResult.xml` and is placed in the working directory.

**XML transform file:** Specifies the output transform file (optional). If not provided, NUnit uses a default transformation.

*Note:* This option does not apply to NUnit v2.5 and later.

**Do not transform output:** If checked, logs raw XML output rather than transforming it. This is useful when debugging problems with the XML format.

**Redirect output to a file:** Specifies a filename to redirect output to instead of logging to the Output pane (optional).

### 3.13.10.4 Options Tab

This tab of the NUnit action configures additional options.

**Processes:** Controls how NUnit loads tests in processes.

**Timeout:** The default timeout to be used for test cases in this run. If any test exceeds the timeout value, it is canceled and reported as an error. If you do not use this option, no timeout is set and tests may run for any amount of time.

**Framework to run under:** Specifies the .NET Framework version to run tests under (optional). When only one version of the CLR is used, the config files for `nunit-console` may be set up to specify that version. As a more convenient alternative when switching CLR, specify the CLR version here.

*Note:* This option requires NUnit v2.2.4 or later.

**Use of AppDomains:** Controls the creation of AppDomains for running tests:  
Default Use multiple domains if multiple assemblies are listed, otherwise a single domain is used.

None No domain is created (the tests are run in the primary domain). This normally requires copying the NUnit assemblies into the same directory as your tests.

Single All tests are run in a single domain. This is how NUnit worked prior to version 2.4.

Multiple A separate test domain is created for each assembly.

e

*Note:* This option requires NUnit v2.4 or later.

**Random seed to generate test cases:** Specifies the seed.

**Number of worker threads:** Number of threads.

**Do not use separate thread:** Suppresses use of a separate thread for running the tests and uses the main thread instead.

### 3.13.10.5 Options (More) Tab

This tab of the NUnit action configures more options.

**No shadow copy:** Disables shadow copying of the assembly in order to provide improved performance.

**Do not convert to short filenames:** If checked, when `clr.bat` is invoked to run tests (the Framework field is provided above), filenames will not be converted to short filenames. NUnit versions prior to v2.4 may require this option to be unchecked, v2.4 and later may require this option to be checked (if "Could not load file or assembly" errors occur).

**Don't save test results:** If checked, don't save any test results.

**Verbose output:** Log additional information as the test runs.

**Internal trace level:** Set the trace level.

**Stop run immediately upon test failure or error:** Whether to stop or continue on failure.

**Show logo:** Displays copyright information at start if checked.

**Additional options:** Used to specify any additional NUnit command-line options.

**NUnit executable:** Overrides the default location of `NUnit-console.exe` (optional). Useful if multiple versions are installed.

### 3.13.11 Sandcastle

This action generates [Sandcastle](#) documentation. Download and install Sandcastle and the Sandcastle Help File Builder before using.

#### Project Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- This action has been tested with the **June 2007** thru **June 2008** releases of *Sandcastle* and **v1.5** thru **v1.7** of *Sandcastle Help File Builder*.
- Use the MSBuild action for *Sandcastle Help File Builder* **v1.8** and later.

#### 3.13.11.1 Project Tab

This tab of the Sandcastle action configures input settings.

**Project or folder:** Specifies the Sandcastle project to document (optional unless Assemblies not specified).

**Assembly/comments files:** Specifies a list of assemblies to document (optional unless Project not specified). Followed by a comma and the name of the comments file (if not named with same root as assembly and .xml extension).

**Property names/values:** Specifies property names and values to pass (optional).

### 3.13.11.2 Options Tab

This tab of the Sandcastle action configures additional options.

**Verbose logging:** If checked, verbose logging will be output.

**Additional options:** Used to specify any additional Sandcastle builder command-line options.

**Sandcastle executable:** Overrides the default location of `SandcastleBuilderConsole.exe` (optional). Useful if multiple versions are installed.

## 3.14 Version Control

### 3.14.1 AccuRev

The AccuRev action creates a step to automate access to the AccuRev version control system. This action provides several screens for configuring repository commands.

When the step is built, this action invokes the AccuRev command-line executable to perform the requested operation.

#### Command Tab

#### Flags Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with AccuRev versions 4 thru 6 and may work with other versions as well. Some fields do not apply to all commands, and some fields are required for certain commands (if `accurev` prompts for input in the build output, one of the required fields is missing). See the *AccuRev User's Guide (CLI Edition)* for more details on the available commands and flags.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.14.1.1 Command Tab

This tab of the AccuRev action configures server and command information.

**Server:** The server containing the depot (optional).

**Port:** The port the server is listening on (optional).

**Command:** The command to perform (required). Most common operations are available in the drop-down list. Other operations may also work but are not explicitly supported. See the *AccuRev User's Guide (CLI Edition)* for more details on the available commands.

**Depot:** The name of the depot to operate on (optional).

**Element names:** One or more elements or to perform the command on, each on a separate line (optional).

**Recursively process directories:** If this option is checked, directories will be recursively processed.

**Version:** The version specification to use for the command (optional).

**Local path:** This specifies the drive+path to place files that are retrieved from or updated in the depot (optional). If the field is left empty, the user's current working directory settings will be used instead (for reliability, especially with automated builds, it is highly recommended that the path be specified here).

**Issue #:** The issue number for the command (optional).

**Username:** The user to use for the command (optional).

### 3.14.1.2 Flags Tab

This tab of the AccuRev action configures additional flags for the command.

**Stream:** The stream to use for the command (optional).

**Transaction range:** Transaction range to operate on (optional).

**Element ID:** Element ID to operate on (optional).

**Select all elements:** Select all elements for the command if checked.

**Element type:** The element type and/or lock type for the command (optional). Separate multiple values with a comma.

**Workspace:** The workspace to use for the command (optional).

**Backing stream:** The backing stream specification to use for the command (optional).

**Location/listing file:** The location or listing file to use for the command (optional).

**Comments:** Specifies comments to be applied to the command (optional).

### 3.14.1.3 Options Tab

This tab of the AccuRev action specifies additional options.

**Output XML format:** Generates XML output if checked.

**EOL type:** Specifies the end-of-line type.

**Reference tree:** The reference tree to use for the command (optional).

**Select all external files/dirs in workspace except ACCUREV\_IGNORE\_ELEMS:** See AccuRev documentation (add command) for more details.

**Include objects even if excluded by ACCUREV\_IGNORE\_ELEMS:** See AccuRev documentation (add command) for more details.

**Convert symbolic link objects in current workspace to symbolic-link elements:** See AccuRev

documentation (add command) for more details.

**Add excluded elements to listing:** See AccuRev documentation (add command) for more details.

**Override:** Override option (details vary depending on command).

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields. See the *AccuRev CLI User's Guide* for more details.

**Specify the command-line executable filename:** If not specified, the action locates `accurev.exe` automatically or expects it to be found in the PATH environment. If the action is unable to locate the command-line tool or multiple versions are installed, enter the full drive+path+filename to `accurev.exe` here.

### 3.14.2 Alienbrain

The Alienbrain action creates a step to automate access to the Avid Technology Alienbrain version control system. This action provides several screens for configuring repository commands.

When the step is built, this action invokes the Alienbrain command-line executable to perform the requested operation.

#### Command Tab

#### Flags Tab

#### Responses Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action requires the Alienbrain client command-line tool to be installed. It has been tested with Alienbrain versions 7 and 8 and may work with other versions as well.

Some fields do not apply to all commands, and some fields are required for certain commands (if ab prompts for input in the build output, one of the required fields is missing). See the Alienbrain The Command Line Tool for more details on the available commands and flags.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.14.2.1 Command Tab

This tab of the Alienbrain action configures server and command information.

**Server:** The server containing the repository (optional).

**Username:** The username to login with (optional).

**Password:** Password of the user (optional).

**Project:** Project to operate on (optional).

**Session ID:** The name of the session to create for the specified logon information, or the ID of a previously-created session to use for this command (optional).

**Command:** The command to perform (required). Most common operations are available in the drop-down list. Other operations may also work but are not explicitly supported. See the Alienbrain The Command Line Tool for more details on the available commands.

**Filenames or directories:** One or more files, directories, or wildcards to perform the command on, each on a separate line (optional).

**Non-recursive:** If this option is checked, directories will not be recursively processed.

**Label:** The label to use for the command (optional).

**Directory for local files:** This specifies the working directory for the command (optional). If the field is left empty, the user's current working directory will be used instead (for reliability, especially with automated builds, it is highly recommended that the path be specified here).

### 3.14.2.2 Flags Tab

This tab of the Alienbrain action configures additional flags for the command.

**Version:** The version to use for the command (optional).

**Branch:** The branch to use for the command (optional).

**Set date/time of local files:** Specifies the timestamp place on files that are retrieved from the server (applies only to checkout, undocheckout, and getlatest commands). *Current* will give each file the current local computer's date/time; *Modification* will set the date/time to match the timestamp of the file version that is retrieved (recommended), and *Checkin* will set the date/time to the date/time when the file was checked in.

**Do not overwrite local copy:** Do not overwrite the local copy if checked.

**Overwrite local writable files:** Whether to replace or skip local files in a get that are writable (applies only to getlatest command).

**Overwrite local checked out files:** Whether to replace or skip local files in a get that are checked out (applies only to getlatest command).

**Ignore existing:** If a file with the same name already exists in the database, then skip this file. If not checked, the import will fail when a file with the same name is found.

**Don't convert newline characters:** If checked, newline characters will not be converted to match those of the target operating system for the get.

**Keep checked out:** If checked, all checkout out files are checked back in and remain checked out.

**Checkout:** The file is automatically checked out after it is imported.

**Exclusive:** If checked, then this will do an exclusive checkout on this item, otherwise a multiple checkout is performed (applies only for checkout command).

**Don't follow symbolic links:** Do not follow any symbolic links in the imported paths.

### 3.14.2.3 Responses Tab

This tab of the Alienbrain action specifies responses to questions asked during command execution.

Use the drop-down list or see the Alienbrain The Command Line Tool help for the list of available response IDs and values. Enter or choose a response ID, enter or chose a response type, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.14.2.4 Options Tab

This tab of the Alienbrain action specifies additional options.

**Parent:** The parent item to use (optional).

**Comments:** Specifies comments to be applied to the command (optional).

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed (optional). Use this to supply flags that are not explicitly supported via the action fields.

**Specify the command-line executable filename:** If not specified, the action locates `ab.exe` automatically. If the action is unable to locate the command-line tool or multiple versions are installed, enter the full drive+path+filename to `ab.exe` here.

## 3.14.3 Bazaar

The Bazaar action automates the Bazaar client. When the step is built, the action invokes the Bazaar command-line executable to perform the requested command.

### Command Tab

### Flags Tab

### Options Tab

### Advanced Tab

### Remote Tab

This action has been tested with Bazaar v2 and may work with other versions as well.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.14.3.1 Command Tab

This tab of the Bazaar action configures information about the command to perform.

**Command:** The command to perform (required). Select a command from the list or enter a command.

**Files:** The targets of the command (optional, one per line). Depending on the command, can specify folders, files, masks, a repository, project, etc.

**Non-recursive:** If this option is checked, the command will be performed non-recursively.



**Revision:** Specifies a revision (optional).

**Working directory:** The working directory for local files (optional).

**Verbose logging:** Self-explanatory.

**Only log errors and warnings:** Self-explanatory.

### 3.14.3.2 Flags Tab

This tab of the Bazaar action configures flags that apply to the command.

**Stacked:** Create stacked branch.

**Standalone:** Do not use shared repository.

**Switch:** Switch checkout in the current directory to the new branch.

**Dry run:** Show what would be done, but don't do anything.

**Subversion:** Subversion repository.

**Create prefix:** Create path leading up to branch if doesn't exist.

### 3.14.3.3 Options Tab

This tab of the Bazaar action configures additional options.

**Message:** Message for the command (optional).

**Author:** Author if different from current user (optional).

**Additional command-line options:** Use this field to enter any additional bzt.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (run bzt.exe for details on the available command-line flags).

**Override the command-line executable filename:** If the action is unable to locate the bzt.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to bzt.exe here.

## 3.14.4 ClearCase

This action creates a step to perform a command an IBM Rational [ClearCase](#) change management server. When the step is built, the action invokes the ClearCase command-line executable to perform the requested action.

### Command Tab

### Options Tab

### Checkout Tab

### Deliver Tab

### Import Tab

**Make Activity Tab**

**Make Element Tab**

**Make Label Tab**

**Make LabelType Tab**

**Make View Tab**

**Make VOB Tab**

**Rebase Tab**

**Set Activity Tab**

**SetCS Tab**

**Update Tab**

**Advanced Tab**

**Remote Tab**

When the step completes, the following temporary macro is created or updated:  
CLEARCASE\_CONFIGSPEC = The configuration specification for the command (catcs command only)

This action has been tested with ClearCase and ClearCase LT 6.0 and may work with other versions as well.

*Note:* See the ClearCase.bld sample for a project utilizing this action.

#### 3.14.4.1 Command Tab

This tab of the ClearCase action configures the source control command to perform and related information.

**Command:** The ClearCase command to perform (required). Most common operations are available in the drop-down list and other commands can be entered as needed. See the [ClearCase documentation](#) for details on the available commands.

**View path:** The local drive and path to use when processing the command (required for most commands unless a Set Current Dir action is used to set the path).

**Pathnames / Object selector:** Identifies one or more file elements, path names, baseline labels, stream selectors or object selectors that the command will operate on (required for most commands).

**Target (or Out or In):** Identifies a pathname that is the target of an ln or mv command.

**Recurse:** Whether to recursively process subfolders.

**Preserve file time:** Determines whether file times are preserved.

**Comment:** A user-provided comment about the command (optional). The comment can be entered directly into the field or the filename of a file containing the comment can be provided.

#### 3.14.4.2 Options Tab

This tab of the ClearCase action configures additional command options.

**Replace:** Replace the existing object with the new object being created. Adds the *-replace* flag to the generated command-line.

**Identical:** Include identical (unchanged) elements. Adds the *-identical* flag to the generated command-line.

**Keep:** Saves the current contents of elements operated on as view-private files. Adds the *-keep* flag to the generated command-line.

**Configuration record only:** Check in the configuration record only. Adds the *-cr* flag to the generated command-line.

**No warn:** Suppresses warning messages. Adds the *-warn* flag to the generated command-line.

**Remove after:** Remove each pname after creating a new version. Adds the *-rm* flag to the generated command-line.

**Local:** Displays information for the local copy of the specified *object-selector*.

**Obsolete:** Include obsolete elements. Adds the *-obsolete* flag to the generated command-line.

**Force:** Suppresses any confirmation steps. Adds the *-force* flag to the generated command-line.

**Additional command-line options:** Use this field to enter any additional ClearCase command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the [ClearCase documentation](#) for details on the available command-line flags).

**Specify the ClearCase executable filename:** If not specified, the action locates expects the ClearCase console application (`cleartool.exe`) to be located in the PATH environment variable. If the action is unable to locate the command-line tool or multiple versions are installed, browse to or enter the full drive+path+filename here.

*Note: the clearfsimport command uses the clearfsimport.exe console application which, if necessary, should instead be supplied in this field when using the clearfsimport command.*

**Convert local paths to UNC:** If checked, relevant local paths will be converted to the equivalent UNC path. This option allows entry of absolute paths using drive letters, the action will convert them automatically to UNC to satisfy the Clear Case requirement for UNC paths (i.e., `C:\Program Files\` becomes `\\<computer name>\C$\Program Files`). Only absolute paths using a drive letter are ever converted, and no paths will be converted when this option is unchecked.

*Note:* This option should be unchecked if you intend to submit absolute paths using drive letters to a Clear Case LT server, for which they are allowed.

#### 3.14.4.3 Checkout Tab

This tab of the ClearCase action configures command options related to a **Checkout** operation.

**Reserved:** If checked the branch is reserved with the checkout. Adds the *-reserved* flag to the generated command-line.

**Unreserved:** If checked the branch is checked out unreserved. Adds the *-unreserved* flag to the generated command-line.

**No master:** Available when **Unreserved** is checked, when checked the branch is checked out even if the current replica does not master the branch. Adds the *-nmaster* flag to the generated command-line.

**No data:** Creates a checkout record for the version, but does not create an editable file that contains its data (not applicable to directories). Adds the *-ndata* to the generated command-line.

**Version:** Allows the checkout of a version that is not the latest on its branch. Adds the *-version* flag to the command-line generated.

**Branch:** Specifies the branch whose most recent version is to be checked out.

#### 3.14.4.4 Deliver Tab

This tab of the ClearCase action configures command options related to a **Deliver** operation.

**Use graphical deliver:** Uses the *-graphical* flag to start the graphical user interface for the deliver operation (don't check for non-interactive builds).

**Complete the deliver operation:** Uses the *-complete* flag to complete the deliver operation.

**Preview only:** Shows which baselines would change and which new activities would be brought into the stream if a rebase operation were to be executed. Adds the *-preview* flag to the generated command-line.

**Abort merge if not fully automatic:** Aborts the operation if a merge is required that can not be completed automatically. Adds the *-abort* flag to the generated command-line.

**Use serial diff output:** Reports differences with each line containing output from one contributor, instead of in a side-by-side format. Adds the *-serial* flag to the generated command-line.

**Perform graphical merge:** Performs a graphical merge for each element that requires it. Adds the *-gmerge* flag to the generated command-line.

**Stream:** Specifies a stream that is the source for the deliver operation.

**Integration-view-tag:** Specifies a view attached to the deliver target stream in the same project or in a different project. is specified with the *-to* flag in the generated command-line.

**Baselines:** Specified a list of baselines to deliver using the *-baselines* flag.

#### 3.14.4.5 Import Tab

This tab of the ClearCase action configures command options related to a **ClearFslImport** operation.

**Downcase:** Uses the *-downcase* flag to change the names of imported elements to lowercase.

**Rmname:** For all source-name arguments that are directories, performs an rmname operation on elements that are already in the VOB but not present in the source directory. Adds the *-rmname* flag to the generated command-line.

**NSetEvent:** Specifies that event records and historical information for new elements and element versions show the user who executed **clearfsimport** and the date of execution, not the original data associated with the sources. Adds the *-nsetevent* flag to the generated command-line.

**Master:** Uses the *-master* flag to assign mastership of the main branch of the element to the VOB replica at which you execute this command.

**Uncheckout:** If a checked-out file element that corresponds to a source file to be imported exists in the target VOB, an **uncheckout** operation is executed on the element and the corresponding view-private file is retained with a suffix of **.keep**. The import operation then checks out the element again and checks in the imported version. Adds the *-unco* flag to the generated command-line.

**Preview:** Preview the import, listing all elements that the import would add or change, as well as any checkouts that would conflict with imports, but does not import anything. Adds the *-preview* flag to the generated command-line.

**Follow:** Processes the object to which a UNIX symbolic link points, instead of importing the link itself into the VOB. Adds the *-follow* flag to the generated command-line.

**Label:** Attaches the specified label instance to each element version checked in. If the corresponding label type does not exist, it is created. Uses the *-mklable* flag in the generated command-line.

#### 3.14.4.6 Make Activity Tab

This tab of the ClearCase action configures command options related to a **Make Activity** operation.

**Headline:** Specifies a description of the activity with the *-headline* flag.

**Stream:** Specifies that the activity be created in this stream using the *-in* flag.

**Don't set as current activity:** Specifies that the new activity not be set as the current activity for the view by adding the *-nset* flag to the generated command-line.

#### 3.14.4.7 Make Element Tab

This tab of the ClearCase action configures command options related to a **Make Element** operation.

**Element type:** Specifies the type of element to be created using the *-eltype* flag.

**No checkout:** Suppressed checkout using the *-nco* flag.

**Checkin:** Creates the new element and version **/main/0**, performs a checkout, and checks in a new version containing the data in view-private file or DO *element-pname*, which must exist. You cannot use this option when creating a directory element. Adds the *-ci* flag to the generated command-line.

**Make path:** For an element that is being created from a view-private file or directory, this option creates elements from its view-private parent directories. Adds the *-mkpath* flag to the generated command-line.

**Master:** Assigns mastership of the **main** branch of the element to the VOB replica in which you execute the **mkelem** command. Adds the *-master* flag to the generated command-line.

#### 3.14.4.8 Make Label Tab

This tab of the ClearCase action configures command options related to a **Make Label** operation.

**Label type:** Specifies the label type for the label, previously created with a **Make LabelType** command.

**Version:** For each *pname*, attaches the label to the version specified by version-selector. Adds the *-version* flag to the generated command-line.

**Config:** Specifies one derived object using the *-config* flag.

**Follow:** For any VOB symbolic link encountered, labels the corresponding target using the *-follow* flag.

#### 3.14.4.9 Make LabelType Tab

This tab of the ClearCase action configures command options related to a **Make LabelType** operation.

**Global:** Creates a label type that can be used as a global resource by client VOBs in the administrative VOB hierarchy. Adds the *-global* flag to the generated command-line.

**Acquire:** When checked, checks all eclipsing types in client VOBs and converts them to local copies of the new global type. Adds the *-acquire* flag to the generated command-line.

**Ordinary:** Uses the *-ordinary* flag to create a label type that can be used only in the current VOB

**Per branch:** Relaxes the default constraint using the *-pbranch* flag, allowing the label type to be used once per branch in a given element's version tree. You cannot attach the same version label to multiple versions on the same branch.

**Shared:** Adds the *-shared* flag, which allows you to create or delete labels of this type at any replica in the VOB family.

#### 3.14.4.10 Make View Tab

This tab of the ClearCase action configures command options related to a **Make View** operation.

**Snapshot:** If specified, uses the *-snapshot* flag to create a snapshot view.

**Tag:** For a dynamic view - specifies a name for the view, in the form of a simple file name. For a snapshot view - specifies a name for the view as it is recorded in the registry.

**Stream:** The view is attached to the specified UCM stream. Uses the *-stream* flag.

**Server storage location:** Specifies a server storage location to hold the view storage directory using the *-stgloc* flag.

**Textmode:** Specifies the textmode for the created view.

#### 3.14.4.11 Make VOB Tab

This tab of the ClearCase action configures command options related to a **Make VOB** operation.

**Tag:** Specifies the tag associated with the created VOB using the *-tag* flag.

**Options:** Specifies mount options to be invoked when the VOB is activated through this VOB tag. Uses the *-options* flag.

**Password:** Specifies a password for the VOB using the *-password* flag.

**UCM project:** If checked the created VOB will be a UCM project VOB for storing UCM related objects. Adds the *-ucmproject* flag to the *-generated* command-line.

**Public:** Uses the *-public* flag to create a public VOB.

#### 3.14.4.12 Rebase Tab

This tab of the ClearCase action configures command options related to a **Rebase** operation.

**Use graphical deliver:** Uses the *-graphical* flag to start the graphical user interface for the deliver operation (don't check for non-interactive builds).

**Complete the deliver operation:** Uses the *-complete* flag to complete the deliver operation.

**Preview only:** Shows which baselines would change and which new activities would be brought into the stream if a rebase operation were to be executed. Adds the *-preview* flag to the generated command-line.

**Abort merge if not fully automatic:** Aborts the operation if a merge is required that can not be completed automatically. Adds the *-abort* flag to the generated command-line.

**Use serial diff output:** Reports differences with each line containing output from one contributor, instead of in a side-by-side format. Adds the *-serial* flag to the generated command-line.

**Perform graphical merge:** Performs a graphical merge for each element that requires it. Adds the *-gmerge* flag to the generated command-line.

**Baselines:** Specifies one or more baselines to use as new foundation baselines for the stream.

**Delete baselines:** Specifies one or more baselines to remove from the stream's configuration.

**View:** Specifies the view in which to execute the rebase command. The view must be associated with a stream that is the stream to be rebased.

**Stream:** Specifies the stream to be rebased.

#### 3.14.4.13 Set Activity Tab

This tab of the ClearCase action configures command options related to a **Set Activity** operation.

**View:** Specifies a view and stream context for the command with a *view-tag* using the *-view* flag.

#### 3.14.4.14 SetCS Tab

This tab of the ClearCase action configures command options related to a **SetCS** operation.

**Kind of change:** specifies the type of change to make to the configuration spec.

- **Default:** Resets the view's config spec to the contents of *default\_config\_spec*, the host's default config spec.
- **Current:** Causes the *view\_server* to flush its caches and reevaluate the current config spec.
- **Stream:** For any UCM view, sets the view's config spec to that defined by the stream it is attached to.
- **Pname:** Specifies a text file whose contents are to become the view's new config spec.
- **Text:** Enter the config spec contents to be used by the SetCS operation.

#### 3.14.4.15 Update Tab

This tab of the ClearCase action configures command options related to an **Update** operation.

**Graphical:** Checking this box will cause the graphical interface to be displayed (not recommended for

non-interactive builds) using the *-graphical* flag.

**Preview only:** Checking this box will use the *-print* flag to log a preview of the update operation without copying or removing files.

**Set current time:** Checking this box sets the time stamp of a file element to the current time using the *-ctime* flag.

**Pname is load rule file:** Specifies that the Pathnames argument (in the **Command** tab) specifies a new load rule file. Adds the *-add\_loadrules* flag to the generated command-line.

**Hijacked files:** Your selection sets the appropriate flag related to the handling of hijacked files (*-overwrite*, *-noverwrite*, *-rename*) in the generated command-line.

**Log file:** Specifies a log file for the operation using the *-log* flag.

### 3.14.5 CVS

This action creates a step to automate use of the [CVS](#) (Concurrent Versions System) version control system. See the CVS.bld sample for sample usage, and view the [CVS manual](#) for more details on the available commands and options.

When the step is built, the CVS command-line executable (cvs.exe) call is constructed based on the inputs and invoked to perform the requested command, and all output is captured to the Output pane and log file if logging is enabled. Some options do not apply to all commands and will be excluded from the generated command-line.

#### Repository Tab

#### Command Tab

#### Global Options Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- CVS version 1.11.2 on Windows does not handle the commit command correctly for local repositories; a newer or older stable release (such as 1.11.1) should be used instead. CVS can be downloaded from the [CVS web site](#) or <ftp://ftp.kinook.com/cvs1.11.1.zip>.
- See the CVS.bld sample for a project utilizing this action.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.14.5.1 Repository Tab

This tab of the CVS action specifies repository login settings.

**Method:** The access method to use when communicating with the repository (required). Pre-defined methods are *ext*, *fork*, *gserver*, *local*, *kserver*, *pserver*, and *sspi*, or a custom access method can be entered. The most common methods are *local* for local repositories and *pserver* for remote servers.

**Host:** The remote server hosting the repository (optional). For instance, *mydomain.com*. Leave blank for local repositories.



**Port:** The port used when communicating with a remote repository (optional).

**Path:** The path to the repository to operate on. For local repositories, this should be the full path ( `\\server\share[\path]` or `<drive>:\path1\[path2...]` ) to the folder containing the `CVSHOME` directory of the repository (and the Browse button can be used to locate the database). For remote repositories, it will be the path to the repository as defined on the server.

**Username:** The username to login with (optional). For remote repositories only.

**Password:** Password of the CVS username in the database (optional).

All of the above inputs are combined into a valid repository string in the form `:method:[user][:password]@hostname[:port]/path/to/repository` (see the [Remote Repositories topic](#) for more details).

### 3.14.5.2 Command Tab

This tab of the CVS action configures the CVS command and its options.

**Command:** The command to perform on the repository (required). Pre-defined commands include `add`, `annotate`, `checkout`, `commit`, `diff`, `export`, `history`, `import`, `init`, `log`, `rdiff`, `release`, `remove`, `rtag`, `status`, `tag`, `update`, and `version`. Other CVS commands can also be entered.

**Files/modules to process:** (Optional) Many of the CVS commands require one or more files or modules (separated by spaces) to operate on. These files or modules are entered in this field. If left blank, all files are matched. For the `rtag` and `tag` commands, this must be the tag name followed by any files (`.` for all files); for the `import` command, enter `repository vendor-tag release-tags` (see the [import sources](#) topic for more details).

**Run only in current working directory:** By default, all commands are processed recursively. To operate only on the root directory specified (non-recursive), check this option.

**Revision or tag to operate on:** (Optional) For many commands, specifies a specific revision or tag to use. If left blank, the most recent revision is used.

*Note:* Normally, prefixing revision field with `D` treats the rest of the value as a date. To prevent this behavior, check the *Do not interpret revision field starting with D as date* option on the global options tab.

**Working directory for local files:** Specifies the local or current directory that will be used when invoking the CVS tool (optional). Where files will be retrieved to or pulled from when processing commands. If specified, the action will also create the folder if it doesn't exist. If left blank, the currently directory of VisBuildPro will be used.

**Log message:** For `import`, `add`, or `commit` commands, specifies a message to be logged in the repository (optional). To specify an empty message value, enter two double quotes (`""`) in this field. If no message value is entered, CVS will open a text editor for providing the message value.

**Additional command options:** Allows for any additional CVS command options to be entered and passed through to  `cvs.exe`  (optional).

### 3.14.5.3 Global Options Tab

This tab of the CVS action specifies global CVS options.

This tab is for specifying global CVS options. Several common options have predefined values, and any other options can be entered in the additional options field (see the [Global Options](#) topic for

details on available global options).

**Quiet mode:** Specifies the *somewhat quiet* mode of CVS.

**Make checked-out files read-only:** Normally, local files are made writeable (CVS works differently than SourceSafe -- the checkout command is more like a VSS Get operation, and local files can be modified at will, and then updated via the commit command). To make local files writeable, check this option.

**Encrypt all communication:** Encrypts all communication between client and server.

**Network traffic compression level:** Valid values are 0 (none) through 9 (highest compression).

**Do not change files:** Only issue reports, do not remove, update, or merge any existing files or create any new files.

**Trace program execution:** Logs additional information while processing a command.

**Do not interpret revision field starting with D as date:** If checked, a revision value on the Command tab starting with a D will not be treated as a date.

**Additional global options:** Allows for any additional CVS global options to be entered and passed through to `cvsexec` (optional).

**Location of `cvsexec`:** (Optional) Defaults to `cvsexec` if blank. If `cvsexec` is located in a path in the PATH environment variable, it will be found and used when this action is built. To specify the full path to the executable (so its location doesn't need to be in the PATH), enter or browse to the full path and filename in this field or define a project or global macro, set it to the full path and filename for `cvsexec`, and reference it here.

### 3.14.6 Dimensions

The Dimensions action creates a step to automate access to the Serena Dimensions version control system. This action provides several screens for configuring commands.

When the step is built, this action invokes the Dimensions command-line executable to perform the requested operation.

#### Server Tab

#### Command Tab

#### Flags Tab

#### Attributes Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with Dimensions CM and Dimensions Express version 2009 R1 and R2 and may work with other versions as well. See the Dimensions Command-Line Reference for more details on the available commands and flags.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.14.6.1 Server Tab

This tab of the Dimensions action configures server information.

**Connection name:** The name of the connection (optional). An existing connection string associated with stored connection parameters (created either by an earlier invocation of this command or by use of the "Previous Connections" field in the remote login dialog box).

**User name:** The operating system user name of your account on the server (optional).

**Password:** The password of your operating system user name account on the server (optional).

**Server:** The host name of the server (optional).

**Database identifier:** The database identifier (optional).

**Data source name:** The data source name for connecting to your remote database (optional).

**Parameter file:** Specifies a file containing the above parameters. The file must be specified using the full directory path. This file has a format similar to the following example:

```
-user dmsys
-pass xxx
-host server1
-dbname intermediate
-dsn PC50
```

### 3.14.6.2 Command Tab

This tab of the Dimensions action configures command settings.

**Command:** The command to perform (optional). Most common operations are available in the drop-down list. See the Dimensions Command-Line Reference for more details on the available commands.

**ID / Name / Spec / Set / Format:** The ID, name, spec, set, format, etc. for the command (optional).

**Status:** Specifies the new status to be given either to the baseline itself or to every item revision in the subset identified above (optional). Unless you hold the PRODUCT-MANAGER role, this status must be reachable from the current status (of each object to be actioned) by a single lifecycle transition. If omitted, the new status is the next normal lifecycle state for each object.

**Working directory:** This specifies the working directory for the command that is executed (optional). If the field is left empty, the user's current working folder settings will be used instead.

**Command file:** Specifies a file containing several Dimensions commands to be executed (optional). The filename must be specified using the full directory path.

### 3.14.6.3 Flags Tab

This tab of the Dimensions action configures additional flags for the command.

**Item filter:** Comprises: <product-id>:<item-id>.<variant>-<item-type>;<revision>

Wildcard characters \_ (underscore) for "any one" and % (percent) for "zero or more" characters may be used to identify what subset of the item revisions in the baseline are to be actioned to a new item lifecycle state.

**Project/stream:** Comprises: <product-id>:<project-id>

This optionally specifies the project/stream to be used for this command: failing this, the user's current project/stream will be taken. *Note:* This command is not available in the Issue Management-only licensed version of Dimensions.

The project/stream is used to select the revision to action if the revision is not actually specified. If the revision is specified, the project or stream is ignored (as Dimensions assumes reference to the explicit revision).

**Baseline:** Specifies a release-baseline which contains the particular revision of <item-spec> to be browsed (optional). It comprises: <product-id>:<baseline-id>

If this qualifier is omitted, the specified or default <revision> is used.

**Part:** Specifies a design part over which this role assignment is applicable (optional).

**Template:** This is the identity of the item-template (optional). This must be:

- Specified when creating a release-baseline
- Omitted when creating a design baseline.

**Type:** Specifies the type of baseline being created (optional). If omitted, the default type is RELEASE if a <template-id> has been specified. (If /TYPE is not specified, then by default, a design baseline is created, which is simply a snapshot of the current stage of product development, and is not therefore expected to need an approval lifecycle).

#### 3.14.6.4 Attributes Tab

This tab of the Dimensions action specifies attributes for the command.

**Attribute names and values:** Enter an attribute name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

#### 3.14.6.5 Options Tab

This tab of the Dimensions action specifies additional options.

**Comment:** A comment for the command (optional).

**Description:** A description for the command (optional).

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields.

**Override the Dimensions executable:** If not specified, the dmcli.exe executable path must be included in the PATH environment variable. If the action is unable to locate the command-line tools or multiple versions are installed, enter the full drive+path to the executable here.

#### 3.14.7 Git

The Git action executes Git version control commands. When the step is built, the action invokes the Git command-line executable to perform the requested command.

## Command Tab

## Options Tab

## Advanced Tab

## Remote Tab

This action has been tested with Git v1 and v2 and may work with other versions as well.

### 3.14.7.1 Command Tab

This tab of the Git action configures information about the command to perform.

**Command:** The command to perform (required).

**Files:** The targets of the command (optional, one per line). Depending on the command, can specify folders, files, masks, a repository, project, etc.

**Revision:** Specifies a revision (optional).

**Working directory:** The working directory for local files (optional).

**Verbose logging:** Self-explanatory.

**Only log errors and warnings:** Self-explanatory.

**Dry run:** Show what would be done, but don't do anything.

### 3.14.7.2 Options Tab

This tab of the Git action configures additional options.

**Message:** Message for the command (optional).

**Author:** Author if different from current user (optional).

**Additional command-line options:** Use this field to enter any additional git.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (run git.exe for details on the available command-line flags).

**Override the command-line executable filename:** If the action is unable to locate the git.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to git.exe here.

## 3.14.8 Mercurial

The Mercurial action executes Mercurial version control commands. When the step is built, the action invokes the Mercurial command-line executable to perform the requested command.

## Command Tab

## Flags Tab

## Options Tab

## Advanced Tab

## Remote Tab

This action has been tested with Mercurial versions 1 thru 3 and may work with other versions as well.

### 3.14.8.1 Command Tab

This tab of the Mercurial action configures information about the command to perform.

**Command:** The command to perform (required).

**Files:** The targets of the command (optional, one per line). Depending on the command, can specify folders, files, masks, a repository, project, etc.

**Repository:** Repository(optional).

**Revision:** Specifies a revision (optional).

**Changeset:** Changeset (optional).

**Working directory:** The working directory for local files (optional).

### 3.14.8.2 Flags Tab

This tab of the Mercurial action configures flags that apply to the command.

**Verbose logging:** Self-explanatory.

**Only log errors and warnings:** Self-explanatory.

**Non-interactive:** Don't prompt and answer yes to all prompts.

**Dry run:** Show what would be done, but don't do anything.

**Include:** Includes (optional).

**Exclude:** Excludes (optional).

### 3.14.8.3 Options Tab

This tab of the Mercurial action configures additional options.

**Message:** Message for the command (optional).

**Additional command-line options:** Use this field to enter any additional hg.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (run hg.exe for details on the available command-line flags).

**Override the command-line executable filename:** If the action is unable to locate the hg.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to hg.exe here.

## 3.14.9 Perforce

The Perforce action creates a step to perform a command on a [Perforce](#) version control depot. It automates much of the initialization of command-line parameters for the Perforce [command-line tool](#) and also adds extra pre- and post-processing and submitting of form information for many actions.

When the step is built, the Perforce command-line executable (P4.exe) call is constructed based on

the inputs and invoked to perform the requested command, and all output is captured to the Output pane and log file if logging is enabled. The path containing `P4.exe` must be included in the PATH environment variable so that it can be located (this is normally done automatically during installation of Perforce). Additional pre- and post- processing also will be performed for many operations (see the comments below for details). Some options do not apply to all commands and will be excluded from the generated command-line. See the `Perforce.bld` sample for sample usage, and view the [Perforce manual](#) for more details on the available commands and options.

## Global Options Tab

## Command Tab

## Change/Form Fields Tab

## Client Spec/Label Tab

## Advanced Tab

## Remote Tab

This action has been tested with versions 2002 thru 2015 of Perforce and may also work with other versions.

### Notes:

- See the `Perforce.bld` sample for a project utilizing this action.
- See the `ContinuousIntegration.bld` sample for a sample of incorporating Perforce into continuous integration builds.

### 3.14.9.1 Global Options Tab

This tab of the Perforce action is for specifying global Perforce options.

**Client:** The client name (optional). Defaults to P4CLIENT environment variable or the computer name if not specified (see [Environment and Registry Variables](#) for details on all environment variables used by Perforce).

**Host:** The host name (optional). Defaults to P4HOST environment variable value if not specified.

**Port:** The server's listen address (optional). If specified, use the format <server name or IP address>:<port number>. Defaults to P4PORT environment variable value if not specified.

**Username:** The username to login with (optional). Defaults to P4USER, USER, or USERNAME environment variable if not provided.

**Password:** Password of the Perforce username in the database (optional). Defaults to P4PASSWD environment variable.

**Working folder:** Specifies a working folder for the command (optional). Overrides any [current directory](#) and replaces it with the specified directory. This can be useful when specifying global settings via a [P4CONFIG](#) file, to specify where to load the configuration file from.

**Additional global options:** Allows for any additional Perforce global options to be entered and passed through to P4 (see [Global Options](#) for details on available global options).

**Don't capture output:** If checked, any P4 output from the step will not be captured or logged (does not apply for change command).

**Don't log details of executed commands:** If checked, the action will not log the details of commands that are executed. This can be useful if the raw P4 output is needed.

### 3.14.9.2 Command Tab

This tab of the Perforce action is for configuring information about the command to be performed.

**Command:** The command to perform on the depot (required). See the [Perforce Command Reference](#) for a description and details on each command's syntax.

#### Special Processing

The Perforce action implements extra pre- and/or post-processing for certain actions:

- For a `change` command, the program output is parsed for the string `Change list 999 created.`, and if found the change list number is stored in the temporary macro **P4\_CHGLIST**.
- The `change`, `submit`, `client`, and `label` commands require supplying a form of inputs related to the command unless the item is being deleted. The Perforce action will add the `-i` command-line option and submit the appropriate form information from the Change List or Client Spec/Label tabs to P4 standard input. This prevents P4 from opening the form in an editor and allow the process to be automated. When using the `change` command to create a changelist or when `submitting` and providing the changelist information, the fields on the Change List tab should be provided and this box checked. When using the `client` or `label` command or to create or update a client specification or label, this box should be checked and the fields on the Client Spec/Label tab provided (not all of the fields are required, see the Perforce documentation for details).
- For a `submit` command, 1) if the *Revert unchanged before submitting* option is checked, before issuing the submit command, a `p4 revert -a` command is performed to revert any files that haven't changed (otherwise Perforce might add a change in the depot for files that have not changed; see the P4 [revert command](#) documentation for details), 2) if the *Folders/Files to process* field is empty, a `p4 opened -c changelist# [files...]` or `p4 opened -c default [files...]` (if no change list is specified) command is issued to retrieve a list of open files in the pending changelist (see the P4 [opened command](#) docs for details); this list is then used to populate the Files list of the submit form (or if the changelist is now empty and the revert option was checked, the changelist is deleted if a changelist number was specified, or the submit action is skipped if the default changelist was specified).

**Folders and/or files to process:** Many of the Perforce commands require one or more files or folders to operate on. The folders or files are entered in this field, one per line. See [File Specifications](#) for details on supported file specifications.

**Change List #/Client/Label:** Specifies a change list number, client name, or label name for the command (optional for most commands).

*Note:* For the [opened](#) command, if this field is empty, the command will operate on the default changelist; to perform an opened command across all changelists, enter `*` in this field.

**Show properties instead of updating:** If checked, the form properties for the given command are logged. If unchecked, the action creates or updates the form properties for that command as specified on the Client Spec/Label or Change/Form Fields tabs.

**Perform delete operation:** Enabled only for commands supporting delete (client, change, fix, and label). If checked, adds the `-d` flag to delete the item rather than submitting a form to create or update the item.



**Revert unchanged before submitting:** Applies only to the submit command; first performs a revert command to discard any unchanged open files.

*Note:* The *Folders and/or files* field must be empty for this option to be enabled.

**Additional command options:** Allows for any additional Perforce command options to be entered and passed through to P4.

### 3.14.9.3 Change/Form Fields Tab

This tab of the Perforce action is used to provide form information for a change or other form command.

**Description:** The description field for the changelist form (required for change command).

**Jobs:** A list of jobs fixed by the changelist, one per line (applies only to change command).

**Additional form fields:** Specifies any additional form fields for a branch, client, change, depot, group, job, jobspec, label, protect, submit, triggers, typemap, or user command, in the following format. See Perforce command reference for details on each command's form fields.

```
# Form comments
#

Field1: single line text

Field2: text

Field3:
  multi
  line
  text

Field4:
  text

Field5: text
```

### 3.14.9.4 Client Spec/Label Tab

This tab is used to provide form information for a client or label command. See the [P4 client](#) and [P4 label](#) for details on the required form fields. Some fields are optional and some are not used for the label command.

**Owner:** The owner of the client

**Descr:** A textual description for the client spec or label.

**Root:** The directory (on the local host) relative to which all files in the view are specified (default is current working directory).

**Options:** Switches that control particular client options (applies only to client command).

**LineEnd:** Controls carriage-return/linefeed (CR/LF) conversion (applies only to client command).

**View:** Specifies the mappings between files in the depot and files in the client workspace (one mapping per line).

### 3.14.10 Plastic SCM

The Plastic SCM action automates the Plastic SCM client. When the step is built, the action invokes the Plastic SCM command-line executable to perform the requested command.

#### Command Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with Plastic SCM v2 thru v5 and may work with other versions as well.

#### 3.14.10.1 Command Tab

This tab of the Plastic SCM action configures information about the command to perform.

**Command:** The command to perform (required).

**Files:** The target of the command (optional, one per line). Depending on the command, can specify folders, files, masks, a repository, project, etc.

**Recursive:** If this option is checked, the command will be performed recursively.

**Working directory:** The working directory for local files (optional).

**Server:** The server to connect to (required for repository commands).

**Port:** The port to connect to (optional, default is 8084).

#### 3.14.10.2 Options Tab

This tab of the Plastic SCM action configures additional options.

**Additional command-line options:** Use this field to enter any additional cm.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (run cm.exe for details on the available command-line flags).

**Override the command-line executable filename:** If the action is unable to locate the cm.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to cm.exe here.

### 3.14.11 PureCM

The PureCM action automates the PureCM client. When the step is built, the action invokes the PureCM command-line executable to perform the requested command.

#### Server Tab

#### Command Tab

#### Flags Tab

#### Options Tab

## Advanced Tab

### Remote Tab

This action has been tested with PureCM 2009 thru 2014 and may work with other versions as well.

#### 3.14.11.1 Server Tab

This tab of the PureCM action configures information about the server to use.

**Server:** The fully qualified machine name or IP address of a running PureCM server (required).

**Port:** The port number of the server (required).

**User name:** The user name used for logging into the server (required).

**Password:** The user's password (required).

#### 3.14.11.2 Command Tab

This tab of the PureCM action configures information about the command to perform.

**Command:** The command to perform (required).

**Files:** The targets of the command (optional, one per line). Depending on the command, can specify folders, files, masks, a repository, project, etc.

**Recursive:** If this option is checked, a command will be performed recursively.

**Changeset:** The base stream changeset (optional).

**Stream:** The stream name (optional).

**Working directory:** The working directory for local files (optional).

**Revision:** A version number of the file specified (optional).

#### 3.14.11.3 Flags Tab

This tab of the PureCM action configures flags that apply to the command.

**Revision:** A revision to process (optional).

**Repository:** Specifies the repository name (optional).

**Source path:** Used with stream merge command.

**Destination path:** Used with the stream merge command.

**Description:** Used with the Share File and Share Project command. If present, will perform a branch operation immediately after the share.

#### 3.14.11.4 Options Tab

This tab of the PureCM action configures additional options.

**Automatically submit:** Automatically submit changes (no prompt).

**Show new name of renamed files:** For changed command.

**Show history in previous streams:** For history command.

**XML output format:** Output in XML format.

**Additional command-line options:** Use this field to enter any additional pcm.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (run pcm.exe for details on the available command-line flags).

**Override the command-line executable filename:** If the action is unable to locate the pcm.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to pcm.exe here.

### 3.14.12 PVCS

The PVCS action creates a step to automate access to the Serena PVCS Version Manager version control system. This action provides several screens for configuring repository commands.

When the step is built, this action invokes one of the PVCS command-line executables to perform the requested operation.

#### Command Tab

#### Flags Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with PVCS Version Manager version 6 and may work with other versions as well. See the PVCS Version Manager Command-Line Reference Guide, Quick Reference Guide, and PCLI User's Guide and Reference for more details on the available commands and flags.

#### Notes:

- See the PVCS.bld sample for a project utilizing this action.
- See the ContinuousIntegration.bld sample for a sample of incorporating PVCS into continuous integration builds.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.14.12.1 Command Tab

This tab of the PVCS action configures command information.

**Command:** The command to perform (required). Most common operations are available in the drop-down list. Other operations may also work but are not explicitly supported. See the *PVCS Version Manager Command-Line Reference Guide*, *Quick Reference Guide*, and *PCLI User's Guide and Reference* for more details on the available commands.

**Repository:** The name of the repository to operate on (optional).

**Workfile and/or archive names:** One or more workfile and/or archive names to perform the command on, each on a separate line (optional). See *Specifying File Names* in the *PVCS Version Manager Command-Line Reference Guide* for more details on the syntax to use.

**Version:** The version to operate on (optional). Enter blank for the current revision, Rrevision for a specific revision, Vlabel for a specific version label, Ddate for a specific date/time stamp, or Gpromo\_group for a specific promotion group. See *Specifying Revisions in Commands* in the *PVCS Version Manager Command-Line Reference Guide* for more details.

**Lock:** Locks the current or specified revision with the current user ID.

**Working directory:** This specifies the working directory for the command that is executed (optional). If the field is left empty, the user's current working folder settings will be used instead.

### 3.14.12.2 Flags Tab

This tab of the PVCS action configures additional flags for the command.

**Answer yes to all questions:** Answer yes in advance to any question the program may ask.

**Check out a writable workfile:** Checks out a writable workfile (does not affect lock status).

**Prevent overwrite of existing writable workfiles:** Denies permission to overwrite existing workfiles.

**Quiet mode:** Selects quiet mode of operation (only reports error conditions).

**Display output:** Displays output but does not create a workfile.

**Diagnostics:** Specifies diagnostic information to log.

**Comment:** Specifies a comment to be applied to the command (optional).

### 3.14.12.3 Options Tab

This tab of the PVCS action specifies additional options.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields.

**Specify the PVCS executable path:** If not specified, the action locates the PVCS command-line tools automatically. If the action is unable to locate the command-line tools or multiple versions are installed, enter the full drive+path to the tools here.

### 3.14.13 SCM Anywhere

Use the SCM Anywhere action to automate the Dynamsoft SCM Anywhere Standard client. This action creates a step for configuring database commands.

When the step is built, the action invokes the SCM Anywhere `scmscmd.exe` command-line executable to perform the requested command. Some options do not apply to all commands and will be excluded from the generated command-line.

#### Database Tab

#### Command Tab

## Flags Tab

## Flags (More) Tab

## Options Tab

## Advanced Tab

## Remote Tab

This action has been tested with SCM Anywhere Standalone v2 and may work with other versions as well.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.14.13.1 Database Tab

This tab of the SCM Anywhere action configures information about the server and database to use.

**Server:** The fully qualified machine name or IP address of a running Dynamsoft server (required).

**Port:** The port number of the server (required).

**User name:** The user name used for logging into the Dynamsoft Server (required).

**Password:** The user's password (required).

**Temporary directory:** Set the local temporary directory (optional). The default temporary directory is the Dynamsoft temporary directory specified in the ServerInfo->General Settings section of the Dynamsoft Server Manager.

### 3.14.13.2 Command Tab

This tab of the SCM Anywhere action configures information about the command to perform.

**Command:** The command to perform (required).

**Working directory:** The working directory where the Command Line Client places a fetched SourceSafe file or project. Also used to specify where on the local machine the Command Line Client looks for a file or project to add or check in to the database.

**Team Project:** The team folder to log into (required for all commands except GetAllTeamProjects).

**Files:** One or more valid files in the specified project (one per line).

**Revision:** A version number of the file specified.

### 3.14.13.3 Flags Tab

This tab of the SCM Anywhere action configures flags that apply to the command.

**Recursive:** If this option is checked, a project level command will be performed recursively.

**Label:** A label that can be associated with a particular project version (optional).

**Permanent:** Used with the Delete command. Permanently deletes a file or project from the database.

**New name:** Used with the Rename command. The new name of a project or file that is being renamed.

**Branch to:** Used with the Share File and Share Project command. If present, will perform a branch operation immediately after the share.

**Share to:** Used with the Share File and Share Project commands. The project into which a file or project will be shared.

**Date range:** Used with the Get File History and Get Project History commands. When used with Get File History, a list of valid date, only required two value, the formats is "datefrom|dateto", the first is date from, the second is date to. If not present, query the data from current time to one year ago. When used with Get Project: only required one value. The parameter can take a variety of formats. For example, the following strings contain acceptable date/time formats: "16 June 2005", "June 16, 2005 6:10:00", "6:10:00 June 16, 2005", "6/16/2005 6:10:00".

**Make local files writable:** Makes the local files writable. Used with Get Latest Files, Get Old Version File, Get File, and Get Project.

#### 3.14.13.4 Flags (More) Tab

This tab of the SCM Anywhere action configures additional flags that apply to the command.

**User:** Used with Get File History. If not specified, the history of all uses is returned.

**Checkin:** Checkin the file.

**Undo:** Undo checking out.

**Keep:** Used with CheckIn File and CheckIn Project. If present, will keep checkout status after the checkin.

**Not read-only:** Used with CheckIn Project, CheckIn File, Undo CheckOut Project, Undo CheckOut File, Add File and Add Folder. If present, will not change the file attribute. If not present, will make file read-only.

**Timestamp of retrieved files:** Specifies the timestamp of files retrieved (optional).

#### 3.14.13.5 Options Tab

This tab of the SCM Anywhere action configures additional options.

**Comment:** Text used for comments (optional).

**Local copy handling:** How to treat local copy of files.

**Additional command-line options:** Use this field to enter any additional scmcmd.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the SCM Anywhere help for details on the available command-line flags).

**Override the executable filename:** If the action is unable to locate the sawscmd.exe command-line tool or multiple versions are installed, enter the full drive+path+filename here.

### 3.14.14 SourceAnywhere

Use the SourceAnywhere action to automate the Dynamsoft SourceAnywhere for VSS and SourceAnywhere Standalone clients. This action creates a step for configuring database commands.

When the step is built, the action invokes the SourceAnywhere `sawvcmd.exe` or `sawscmd.exe` command-line executable to perform the requested command. Some options do not apply to all commands and will be excluded from the generated command-line.

#### Database Tab

#### Command Tab

#### Flags Tab

#### Flags (More) Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

##### Notes:

- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.
- This action has been tested with SourceAnywhere for VSS v5 and v6 and SourceAnywhere Standalone v2 thru v6 and may work with other versions as well. Some commands and options apply only to the VSS or Standalone product.

#### 3.14.14.1 Database Tab

This tab of the SourceAnywhere action configures information about the server and database to use.

**Server:** The fully qualified machine name or IP address of a running Dynamsoft server (required).

**Port:** The port number of the server (required). Default port is 8787 for SourceAnywhere for VSS and 7777 for SourceAnywhere Standalone.

**User name:** The user name used for logging into the Dynamsoft Server (required).

**Password:** The user's password (required).

**Alias:** The alias that the Dynamsoft Server has defined to point to a SourceSafe database (optional, applies to SourceAnywhere for VSS only). This name is specified under the Databases section of the Dynamsoft Server Manager.

**Temporary directory:** Set the local temporary directory (optional). The default temporary directory is the Dynamsoft temporary directory specified in the ServerInfo->General Settings section of the Dynamsoft Server Manager.

**Verbose logging:** Outputs verbose status messages, such as connections, server responses, etc.



### 3.14.14.2 Command Tab

This tab of the SourceAnywhere action configures information about the command to perform.

**Command:** The command to perform (required).

**Working directory:** The working directory where the Command Line Client places a fetched SourceSafe file or project. Also used to specify where on the local machine the Command Line Client looks for a file or project to add or check in to the database.

**Repository:** The repository to operate on (applies only to SourceAnywhere Standalone).

**Project:** The project to operate on (required for all commands except GetAllTeamProjects).

**Files:** One or more valid files in the specified project (one per line).

**Revision:** A version number of the file specified.

### 3.14.14.3 Flags Tab

This tab of the SourceAnywhere action configures flags that apply to the command.

**Recursive:** If this option is checked, a project level command will be performed recursively.

**Label:** A label that can be associated with a particular project version (optional).

**Permanent:** Used with the Delete command. Permanently deletes a file or project from the database.

**New name:** Used with the Rename command. The new name of a project or file that is being renamed.

**Branch:** Used with the Share File and Share Project command. If present, will perform a branch operation immediately after the share.

**Share to:** Used with the Share File and Share Project commands. The project into which a file or project will be shared.

**Date range:** Used with the Get File History and Get Project History commands. When used with Get File History, a list of valid date, only required two value, the formats is "datefrom|dateto", the first is date from, the second is date to. If not present, query the data from current time to one year ago. When used with Get Project: only required one value. The parameter can take a variety of formats. For example, the following strings contain acceptable date/time formats: "16 June 2005", "June 16, 2005 6:10:00", "6:10:00 June 16, 2005", "6/16/2005 6:10:00".

**Make local files writable:** Makes the local files writable. Used with Get Latest Files, Get Old Version File, Get File, and Get Project.

### 3.14.14.4 Flags (More) Tab

This tab of the SourceAnywhere action configures additional flags that apply to the command.

**User:** Used with Get File History. If not specified, the history of all uses is returned.

**Checkin:** Checkin the file.

**Undo:** Undo checking out.

**Keep:** Used with CheckIn File and CheckIn Project. If present, will keep checkout status after the

checkin.

**Include file history:** Used with Show Project History. If present, will show file history together with project history.

**Not read-only:** Used with CheckIn Project, CheckIn File, Undo CheckOut Project, Undo CheckOut File, Add File and Add Folder. If present, will not change the file attribute. If not present, will make file read-only.

**No compression:** Turns off compression.

**Timestamp of retrieved files:** Specifies the timestamp of files retrieved (optional).

**End-of-line terminator:** Specifies line terminators used in text files (optional).

#### 3.14.14.5 Options Tab

This tab of the SourceAnywhere action configures additional options.

**Comment:** Text used for comments (optional).

**Local copy handling:** How to treat local copy of files.

**File type:** Type of file.

**Additional command-line options:** Use this field to enter any additional sawvcmd.exe or sawscmd.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the SourceAnywhere help for details on the available command-line flags).

**Override the executable filename:** If the action is unable to locate the sawvcmd.exe/sawscmd.exe command-line tool or multiple versions are installed, enter the full drive+path+filename here.

#### 3.14.15 SourceOffSite

Visual Build provides the SourceOffSite action to automate the SourceGear SourceOffSite client. This action creates a step for configuring database commands.

When the step is built, the action invokes the SourceOffSite `soscmd.exe` command-line executable to perform the requested command. Some options do not apply to all commands and will be excluded from the generated command-line.

##### Database Tab

##### Command Tab

##### Flags Tab

##### Options Tab

##### Advanced Tab

##### Remote Tab

This action has been tested with SourceOffSite v4 and v5 and may work with other versions as well.

*Note:* By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.14.15.1 Database Tab

This tab of the SourceOffSite action configures information about the server and database to use.

**Server:** The fully qualified machine name or IP address of a running SourceOffSite server (required).

**Port:** The port number of the server (required).

**Database:** A valid SourceSafe database being served by the SourceOffSite server (Database or Alias required for most commands).

**Alias:** A valid name that the SOS Server has defined to point to a configured SourceSafe database (optional). This name is specified under the Databases tab of the SourceOffSite Server Manager utility.

**User name:** The SourceSafe user name used for logging into the SourceOffSite server (required).

**Password:** The SourceSafe user's password (required).

**Verbose logging:** Outputs verbose status messages, such as connections, server responses, etc.

### 3.14.15.2 Command Tab

This tab of the SourceOffSite action configures information about the command to perform.

**Command:** The command to perform (required).

**Working directory:** The working directory where the Command Line Client places a fetched SourceSafe file or project. Also used to specify where on the local machine the Command Line Client looks for a file or project to add or check in to the database.

**Project:** A valid SourceSafe project (folder) in the specified database.

**Files:** One or more valid SourceSafe files in the specified project (one per line).

**Revision:** A version number of the file specified.

**Home directory:** The path to the directory where SourceOffSite stores user specific data (if the SourceOffSite GUI Client is installed). By default, this directory is `C:\Documents and Settings\\Application Data\SourceGear\SOS`.

### 3.14.15.3 Flags Tab

This tab of the SourceOffSite action configures flags that apply to the command.

**Label:** A label that can be associated with a particular project version (optional).

**Recursive:** If this option is checked, a project level command will be performed recursively.

**Permanent:** Used with the Delete command. Permanently deletes a file or project from the database.

**Skip writable:** Used with Get Project command. If checked, prevents checked out files or other writable files from being overwritten.

**New name:** Used with the Rename command. The new name of a project or file that is being renamed.

**Branch:** Used with the Share File and Share Project command. If present, will perform a branch operation immediately after the share.

**Share to:** Used with the Share File and Share Project commands. The project into which a file or project will be shared.

**No cache:** If checked, prevents the Command Line Client from using the SourceOffSite cache file (databaseX.sos) when retrieving files. Use of this parameter will force the Command Line Client to retrieve ALL files, regardless of whether the files have changed or not and retrieve files to the directory from which the soscmd command was issued (unless a working directory is specified).

**No compression:** Turns off compression.

#### 3.14.15.4 Options Tab

This tab of the SourceOffSite action configures additional options.

**Comment:** Text used for comments (optional).

**Timestamp of retrieved files:** Specifies the timestamp of files retrieved (optional).

**End-of-line terminator:** Specifies line terminators used in text files (optional).

**Use old return status codes:** Use the "old" return status codes. The Command Line Client now returns 0 on success, -1 on failure.

**Additional command-line options:** Use this field to enter any additional soscmd.exe command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see soscmd.txt for details on the available command-line flags).

**Override the soscmd.exe executable filename:** If the action is unable to locate the soscmd.exe command-line tool or multiple versions are installed, enter the full drive+path+filename to soscmd.exe here.

#### 3.14.16 Subversion

This action creates a step to automate use of the [Subversion](#) version control system. See the SVN.bld sample for example usage, and view the [Subversion book](#) for more details on the available subcommands and options.

When the step is built, the Subversion command-line executable (`svn.exe`, `svnadmin.exe`, or `svnlook.exe`) call is constructed based on the inputs, invoked to perform the requested subcommand, and all output is captured to the Output pane and log file if file logging is enabled.

##### Repository Tab

##### Subcommand Tab

##### Switches Tab

##### Options Tab

##### Advanced Tab

## Remote Tab

This action has been tested with Subversion versions 1.0 - 1.9 and may also work with other versions.

*Notes:*

- See the Subversion.bld sample for a project utilizing this action.
- See the ContinuousIntegration.bld sample for a sample of incorporating Subversion into continuous integration builds.

### 3.14.16.1 Repository Tab

This tab of the Subversion action is used to specify information for connecting to the [Subversion](#) repository and providing credentials to the server.

**Protocol:** The protocol to use when communicating with the repository host (optional). Pre-defined methods are file (local), svn, svn+ssh, http, and https, or a custom access method can be entered. If left blank, file will be assumed.

**Host:** The remote server hosting the repository (optional). For instance, mydomain.com. Leave blank for local repositories.

**Port:** The port used when communicating with a remote repository (optional).

**Path:** The base path to the repository to operate on. For local repositories, this should be the full path (`\\server\share[\path]` or `<drive>:\path1\[path2. . .]`) to the folder containing the root directory of the repository (and the Browse button can be used to locate the database). For remote repositories, it will be the path to the repository as defined on the server.

All of the above inputs are combined into a valid repository string in the form `protocol://hostname[:port]/path/to/repository` (see the [Repository Administration](#) and [Server Configuration](#) chapters of the Subversion book for more details) and are prefixed to each filename or path provided in the Files field on the Subcommand tab.

**Username:** The username to login with (optional). For remote repositories only.

**Password:** Password of the Subversion username in the database (optional).

**Config dir:** Specifies a directory to read configuration information from instead of the .subversion subdirectory in the user's home directory (optional).

**Enable auto-props:** Enables auto-props, overriding the enable-auto-props directive in the config file.

**Disable auto-props:** Disables auto-props, overriding the enable-auto-props directive in the config file.

**Don't cache authentication information:** Prevents caching of authentication information (e.g., username and password) in the Subversion administrative directories.

**Force suspicious parameters to be accepted:** Forces a suspicious parameter passed to the Message or File options to be accepted as valid. By default, Subversion will produce an error if parameters to these options look like they might instead be targets of the subcommand.

### 3.14.16.2 Subcommand Tab

This tab of the Subversion action is used to provide information about the Subversion [subcommand](#) to be performed.

**Subcommand:** The subcommand to perform on the repository (required). Select or enter the subcommand to use. See the [Subversion Complete Reference](#) for a description and details on available subcommands.

**Executable:** The Subversion command-line executable to invoke. Subversion provides multiple command-line programs, each providing different functionality. The subcommand list will be populated based on the subcommands available for the selected executable.

**Files/modules to process:** Many of the Subversion subcommands require one or more files or modules (one per line) to operate on (optional). These files or modules are entered in this field. If left blank, all files are matched.

**Run only in current directory:** By default, all subcommands are processed recursively. To operate only on the root directory specified (non-recursive), check this option.

**Property name:** The name of the property to operate on (optional). Applies only for prop\* commands.

**Revision to operate on:** For many subcommands, specifies a specific revision or tag to use (optional). If left blank, the most recent revision is used. See the SVN book's [Revisions topic](#) for valid revision strings.

**Operate on revision property instead of file- or directory-specific property:** Operates on a revision property instead of a property specific to a file or directory. This option requires that you also specify a Revision in the previous field.

**Working directory for local files:** Specifies the local or current directory that will be used when invoking Subversion (optional). Where files will be retrieved to or pulled from when processing subcommands. If specified, the action will also create the folder if it doesn't exist. If left blank, the currently directory of VisBuildPro will be used.

**Log message:** Specifies a commit message to be logged in the repository (optional).

### 3.14.16.3 Switches Tab

This tab of the Subversion action is for specifying global SVN switches.

**Quiet mode:** Requests that the client print only essential information while performing an operation.

**Dry run:** Goes through all the motions of running a command, but makes no actual changes.

**Force command to run:** Forces a particular command or operation to run.

**XML output format:** Prints output in XML format.

**Verbose mode:** Requests that the client print out as much information as it can while running any subcommand.

**Non-interactive mode:** If checked, the step will fail instead of prompting for missing inputs (applies only for svn.exe).

**Recursive:** Forces subcommand to operate recursively.

**Keep changelist after commit:** Don't delete changelist after commit.

**Keep local copy of file or directory after delete:** Keep the local copy of a file or directory (used with the delete subcommand).

**Don't print differences for deleted files:** Prevents Subversion from printing differences for deleted files. The default behavior when you remove a file is for the diff subcommand to print the same differences that you would see if you had left the file but removed all the content.

**Show omitted files:** Shows files in the status listing that would normally be omitted since they match a pattern in the global-ignores configuration option or the svn:ignore property.

**Don't automatically unlock files:** Don't automatically unlock files (the default commit behavior is to unlock all files listed as part of the commit).

**Create and add non-existent and non-versioned parent subdirectories:** Create and add non-existent or non-versioned parent subdirectories to the working copy or repository as part of an operation. This is useful for automatically creating multiple subdirectories where none currently exist. If performed on a URL, all the directories will be created in a single commit.

**Ignore external definitions:** Tells Subversion to ignore external definitions and the external working copies managed by them.

#### 3.14.16.4 Options Tab

This tab of the Subversion action is for specifying additional global options. Several common options have predefined values, and any other options can be entered in the additional options field (see the [svn Options](#) topic for details on available global options).

**Ignore ancestry when calculating differences:** Tells Subversion to ignore ancestry when calculating differences (rely on path contents alone).

**Pay attention to ancestry when calculating differences:** Pay attention to ancestry when calculating differences.

**Incremental output:** Prints output in a format suitable for concatenation.

**Stop harvesting information when a copy is encountered:** Causes a Subversion subcommand which is traversing the history of a versioned resource to stop harvesting that historical information when a copy--that is, a location in history where that resource was copied from another location in the repository--is encountered.

**Changelists:** Operate only on members of the specified changelists (optional).

**Max messages:** Show only the first X log messages..

**Additional switches:** Allows for any additional Subversion switches (optional). See the [Subversion book](#) for details on available switches.

**Location of Subversion command-line tools:** The path containing the Subversion command-line executables (optional). If the executable is located in a path in the PATH environment variable, this field can be blank. To specify the full path to the executable (so its location doesn't need to be in the PATH), enter or browse to the full path and filename in this field or define a project or global macro, set it to the full path and filename for `svn.exe`, and reference it here.

### 3.14.17 Surround SCM

The Surround SCM action creates a step to automate access to the Seapine [Surround SCM](#) version control system. This action provides several screens for configuring repository commands. See SurroundSCM.bld for sample usage.

When the step is built, this action invokes the Surround SCM command-line client executable to perform the requested operation.

#### Surround SCM Tab

#### Flags Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with Surround SCM versions 1.2 thru 2015 and may work with other versions as well. Some fields do not apply to all commands, and some fields are required for certain commands (if sscm prompts for input in the build output, one of the required fields is missing). See the Surround SCM CLI Reference Guide for more details on the available commands and flags.

*Notes:*

- See the SurroundSCM.bld sample for a project utilizing this action.
- See the ContinuousIntegration.bld sample for a sample of incorporating Surround SCM into continuous integration builds.

#### 3.14.17.1 Surround SCM Tab

This tab of the Surround SCM action configures server and repository information.

**Server:** The server containing the repository (optional).

**Port:** The port the server is listening on (optional, default 4900).

**Username:** The username to login with (optional).

**Password:** Password of the user in the repository (optional).

**Command:** The command to perform (required). Most common operations are available in the drop-down list. Other operations may also work but are not explicitly supported. See the Surround SCM CLI Guide for details on the available operations.

**Repository:** The name of the repository to operate on (optional).

**Items or files to process:** One or more items, folders, masks, or filenames to perform the command on, each on a separate line (optional).

**Directory for local files:** This specifies the drive+path to place files that are retrieved from or updated in the database (optional). If the field is left empty, the user's current working folder settings will be used instead (for reliability, especially with automated builds, it is highly recommended that the path be specified here).



**Label:** The label to use for the command (optional).

### 3.14.17.2 Flags Tab

This tab of the Surround SCM action configures additional flags for the command.

**Version:** The version to use for the command (optional).

**Branch:** The branch to use for the command (optional).

**Recursively process files and sub-repositories:** If this option is checked, files will be recursively processed.

**Quiet mode:** Do not list repository and local full path of files if checked.

**Make local files writeable:** Make local files editable if checked.

**Force file retrieval:** Force file retrieval from the server regardless of local copy status.

**Comment:** Specifies a comment to be applied to the command (optional).

### 3.14.17.3 Options Tab

This tab of the Surround SCM action specifies additional options.

**Handling of writeable local files:** How to handle a local writeable file.

**Set date/time of local files:** Specifies the timestamp placed on files retrieved from the repository.

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields (see the Surround SCM CLI Reference Guide for more details on the available commands and flags).

**Specify the command-line executable filename:** If not specified, the action locates `sscm.exe` by looking for the default registry value `HKEY_LOCAL_MACHINE\SOFTWARE\Seapine Software\Surround SCM\Directory` and combines its path with `sscm.exe`. If the action is unable to locate the command-line tool or multiple versions are installed, check the checkbox above this field and enter the full drive+path+filename to `sscm.exe` here.

### 3.14.18 Team Concert

The Team Concert action creates a step to execute commands in the IBM Rational Team Concert / Jazz source control system. When the step is built, this action invokes the Team Concert command-line executable to perform the requested operation.

#### Server Tab

#### Command Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

This action has been tested with Team Concert v3 thru v4 and may work with other versions as well. See the *Using the command-line client* in the Rational Team Concert help for more details on the available commands and flags.

*Note:* Rational Team Concert and the Client for Eclipse IDE must be installed in order for the command-line client to be available (command line tools are installed in the IBM/TeamConcert/scmtools/eclipse folder of the Team Concert installation by default).

#### 3.14.18.1 Server Tab

This tab of the Team Concert action configures server information.

**Repository:** Specifies the repository in which to take this action (optional). If you have stored repository credentials using the login command, you can use the nickname you supplied for these stored credentials

**User name:** Specifies a user ID that exists in the named repository (optional). If you specified a repository URI or nickname for which you have stored credentials using the login command, this option is ignored.

**Password:** The password for the specified user ID in the named repository (optional). If you specified a repository URI or nickname for which you have stored credentials using the login command, this option is ignored.

**Workspace:** Workspace containing the change set to close (optional). Applies only to the changeset close command.

#### 3.14.18.2 Command Tab

This tab of the Team Concert action configures command settings.

**Command:** The command to perform (optional). Most common operations are available in the drop-down list.

**Path names / change sets / components:** The path names, change sets, or components for the command (optional, one per line).

**Workspace path:** The path name of a local workspace created by the load or share command (optional). You can omit this option if the current working directory is in a local workspace.

#### 3.14.18.3 Options Tab

This tab of the Team Concert action specifies additional options.

**Suppress informational messages:** Suppress most informational messages while operating.

**Verbose output:** A description for the command (optional).

**Overwrite existing files when loading:** Overwrite existing files when loading (only applies to load command).

**Resolve conflicts using the currently checked-in version:** Resolve conflicts using the currently checked-in version (only applies to resolve command).

**Description:** A description for the command (optional, only applies to create commands).

**Additional command-line options:** Use this field to enter any additional command-line options to be

added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields.

**Override the Team Concert executable:** If not specified, the location of scm.exe must be included in the PATH environment variable. If the action is unable to locate the command-line tools or multiple versions are installed, enter the full drive+path to the executable here.

### 3.14.19 Vault / Fortress

The Vault / Fortress actions create a step to automate access to the SourceGear [Vault](#) / Vault Professional / Fortress version control / bug tracking system. This action provides several screens for configuring repository commands. See Vault.bld for sample usage.

When the step is built, this action invokes the Vault / Fortress command-line client executable to perform the requested operation and captures its output.

#### Server Tab

#### Command Tab

#### Flags Tab

#### Options and Version 1 Tabs

#### Advanced Tab

#### Remote Tab

This action has been tested with Vault versions 1 through 8 and Fortress versions 1 and 2 and may work with other versions as well. For more details on the available commands and flags, see the Vault / Fortress documentation or create a Vault / Fortress action with a command of help (leave the Repository field blank to get a list of all commands, or for Vault versions 2.0+, enter the command to get help on in the Repository field).

#### Notes:

- See the Vault.bld sample for a project utilizing this action.
- See the ContinuousIntegration.bld sample for a sample of incorporating Vault / Fortress into continuous integration builds.
- By default, the Fortress action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.14.19.1 Server Tab

This tab of the Vault action configures server information.

**Host:** The server containing the Vault / Fortress repository (required for most commands).

**Username:** The username to login with (required for most commands).

**Password:** Password of the user in the repository (optional).

**Connect using SSL:** Check this option to use secure communication with the server.

**Proxy Server:** Proxy server to connect to (optional).

**Proxy Port:** Proxy port to connect on (optional).

**Proxy Domain:** Proxy domain (optional).

**Proxy User:** Proxy username (optional).

**Proxy Password:** Password of the proxy user (optional).

### 3.14.19.2 Command Tab

This tab of the Vault action configures repository and command information.

**Command:** The Vault / Fortress command to perform (required). Most common operations are available in the drop-down list. Other operations may also work but are not explicitly supported.

*Note:* For the *branch*, *commit*, *delete*, *get*, *label*, and *share* commands, enter the target repository folder or additional repository paths as the first parameters of the *Additional command-line options* field on the Options tab.

**Repository:** The name of the Vault / Fortress repository to operate on (required for most commands).

**Repository paths or files to process:** One or more Vault / Fortress paths, masks, or filenames to perform the command on, each on a separate line (optional). Folder names are prefixed with \$ and use slashes rather than backslashes for folder delimiters.

*Note:* For the diff command, enter the both objects to compare in this field, each on a separate line; for the undochangesetitem command, enter the changeset item id here.

**Path for local files/destination folder:** This specifies the drive+path to place files that are retrieved from the database (optional). If the field is left empty, the user's current working folder settings will be used instead (for reliability, especially with automated builds, it is highly recommended that the path be specified here).

*Note:* For the batch command, enter the filename containing the batch commands here; for the move, rename, and share commands, enter the destination repository folder here.

**Version/Wildcard/Label/Line Number:** The version to use for pin and getversion commands (optional), the wildcard to use for the getwildcard command, the label to get on for the getlabel and label commands, or the line number to search for the blame command.

### 3.14.19.3 Flags Tab

This tab of the Vault action specifies additional flags about the command.

**Flags that apply only for get, getlabel, getversion, and getwildcard commands of Vault 2.x and later**

**Get all folders regardless of cloak status:** Ignores folder cloak status. Applies only for get, getversion, and getwildcard commands.

**Make files writeable:** Make all files writeable after retrieval.

**Make files read-only:** Make all files read-only after retrieval.

**Other flags**

**Do not process folders recursively:** If this option is checked, only the specified folder(s) will be

processed and not any subfolders. This option only applies to the diff, get, getlabel, getversion, getwildcard, listfolder, and undocheckout commands.

**Set date/time of local files:** Specifies the timestamp placed on files that are retrieved from the repository. *Current* will give each file the current local computer's date/time; *Modification* will set the date/time to match the timestamp of the file version that is retrieved (recommended), and *Checkin* will set the date/time to the date/time when the file was checked into Vault. This flag applies only for the get, getlabel, getversion, getwildcard, and checkout commands.

**Merge behavior:** Specifies the action to take when updating a local file with new content. *Automatic* will attempt to merge changes from the server (applies only to get and getwildcard commands); *Later* will not overwrite existing, modified files, and *Overwrite* will overwrite the local file with the server's file. This option applies only for the checkout, get, getlabel, getversion, getwildcard, and checkout commands.

**Commit unchanged action:** Specifies the action to perform on an unchanged, checked out file during commit. *Checkin* will check the file in unmodified; *Leave* will leave the file checked out; and *Undo checkout* will undo the checkout of the file. This option applies only for the commit command.

**Keep checked out:** All files remain checked out upon commit (applies only to commit command).

**Exclusive checkout:** Checks out items exclusively (applies only to checkout command).

**Commit changes:** By default, the following commands append the specified actions to the pending change set: add, createfolder, delete, move, pin, rename, share, unpin. Items in the pending change set do not take effect until the commit command is used to commit them to the server. However, each of the above commands accepts the -commit option, which causes that change to be committed immediately.

#### 3.14.19.4 Options and Version 1 Tabs

This tab of the Vault action configures additional options.

**Comments:** Specifies a comment to be applied to the command (optional).

**Additional command-line options:** Use this field to enter any additional command-line options to be added to the call that is constructed. Use this to supply flags that are not explicitly supported via the action fields. For more details on the available flags, see the [help page](#) on the SourceGear web site or create a Vault / Fortress action with a command of help (leave the Repository field blank to get a list of all commands, or for Vault v2 and later, enter the command to get help on in the Repository field).

**Specify the command-line executable filename:** If not specified, the action locates the Vault / Fortress command-line executable. If the action is unable to locate the command-line tool or multiple versions are installed, check the checkbox above this field and enter the full drive+path+filename to vault.exe here.

### Version 1 Tab

#### *Flags that apply only for 'get' command of Vault v1*

**Overwrite all files regardless of comparison status:** Overwrite all local files with the Vault version.

**Don't create sgvault folders:** Don't create \_sgvault folders.

**Verbose mode:** Print a list of every file retrieved. This can be slow.

## 3.15 Virtual Machines

### 3.15.1 Hyper-V

The Hyper-V action performs an operation on a Microsoft Hyper-V virtual machine. See here for options for automating build steps within a virtual machine.

#### Hyper-V Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

##### Notes:

- This action has been tested with Windows 8, Windows Server 2012 / 2012 R2, Microsoft Hyper-V Server 2008 / 2008 R2, and Windows Server 2008 / 2008 R2 with Hyper-V and may work with other versions as well.
- PowerShell v2 or later (included with Windows 7 and Windows Server 2008 R2 and later) and the PowerShell Management Library for Hyper-V module (included with Windows 8 and Windows Server 2012 and later) must be installed on the machine where Visual Build is installed.

#### 3.15.1.1 Hyper-V Tab

This tab of the Hyper-V action specifies the operation and machine information.

**Command:** The virtual machine command to perform.

**Server:** The name of the Hyper-V server machine (optional, blank for local computer).

**Virtual machine:** The name of the virtual machine, or filename when registering a machine (required). For operations other than Register, \* in this field indicates to perform the operation on all virtual machines on the given server, and a regular expression can also be entered in this field to perform the operation on all machine names matching the regular expression.

**Parameter names/values:** Specifies the names and values of parameters for the command being executed. See the Hyper-V Cmdlets documentation for each command's parameters.

#### 3.15.1.2 Options Tab

This tab of the Hyper-V action configures additional options.

**Seconds to wait for response from heartbeat integration component:** If specified, the VM is checked for a response from the Heartbeat integration component every 5 seconds until one is found or the timeout expires (optional).

**Wait for job to complete:** If checked, specifies that the command should not return until the WMI job completes.

**Do not prompt:** Ensures that no prompting occurs before the action is carried out.

**Additional options:** Specified additional parameters for the PowerShell Management Library for Hyper-V module (optional). See the documentation for available parameters.

**PowerShell Management Library for Hyper-V filename:** Specifies the path and filename to the

PowerShell Management Library for Hyper-V module (HyperV.psd1 or Hyper-V.psd1). If the module is already included in the PowerShell module path (Windows 8 and Windows 2012), clear out this field value to prevent explicit loading of the module.

**Override PowerShell executable filename:** Overrides the default location for the PowerShell executable (optional).

*Note:* To call the 64-bit version of PowerShell from the 32-bit edition of Visual Build, enter a value of %WINDIR%\Sysnative\WindowsPowerShell\v1.0\powershell.exe (replacing v1.0 with the desired version) in this field. To call the 32-bit version of PowerShell from the 64-bit edition of Visual Build, enter a value of %WINDIR%\SysWOW64\WindowsPowerShell\v1.0\powershell.exe (replacing v1.0 with the desired version) in this field.

### 3.15.2 Parallels

This action creates a step to perform an operation on a Parallels Desktop virtual machine.

**Command:** The virtual machine command to perform.

**Virtual machine:** The name or filename of the virtual machine.

**Additional command-line options:** Adds the specified options to the Parallels command line (optional). See the Parallels Command Line Reference Guide for valid options.

**Override Parallels executable filename:** Overrides the default location for the Parallels executable (optional).

*Notes:*

- This action has been tested with Parallels v4 and v6 and may work with other versions as well.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

### 3.15.3 Virtual PC

The Virtual PC action performs an operation on a Microsoft Virtual PC virtual machine. See here for options for automating build steps within a virtual machine.

#### Virtual PC Tab

#### Options Tab

#### Advanced Tab

#### Remote Tab

*Notes:*

- This action has been tested with Virtual PC 2004 and 2007.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.15.3.1 Virtual PC tab

This tab of the Virtual PC action specifies the operation and machine information.

**Operation:** The virtual machine operation to perform.

**Machine name:** The name or filename of the virtual machine.

**Don't start Virtual PC console:** Starts the specified virtual machine without starting Virtual PC Console. This can be useful for simplifying the Virtual PC interface in a corporate or lab environment, where users only want to access a virtual machine and do not need access to Virtual PC Console. Applies only when starting Virtual PC.

**Don't start machines configured to start automatically:** Prevents the starting of any virtual machines that are configured to start automatically. Applies only when starting Virtual PC.

**Turn on host-side disk caching:** Turns on host-side disk caching, which can improve performance of virtual machines running operating systems other than Windows. This can be useful for resolving poor performance problems with disk intensive tasks. Applies only when starting Virtual PC.

**Disable close button:** Disables the Close button on the specified virtual machine. Applies only when starting the virtual machine.

**Clip accelerated S3 bit coordinates to 12 bits:** Clips accelerated S3 bit coordinates to 12 bits on the specified virtual machine. Applies only when starting the virtual machine.

**Network connections operate in external only mode:** Specifies that all network connections on the specified virtual machine operate in external only mode. This limits the access of the virtual machine to only resources external to the physical computer. Applies only when starting the virtual machine.

**Screen mode:** Optionally changes the screen mode between full screen, windowed, and minimized.

**Bring window to foreground:** Brings the specified virtual machine to the foreground of the screen. Applies only when the virtual machine is running.

### 3.15.3.2 Options tab

This tab of the Virtual PC action configures additional options.

**Disable direct execution mode:** Disables any direct mode execution within the specified virtual machine. This can be useful for providing a temporary resolution for poor stability of applications on a guest operating system. Applies only when starting the virtual machine.

**BIOS serial number:** Sets the BIOS serial number of the specified virtual machine to the value specified (optional). This can be useful for tracking Virtual PCs when using hardware management software. Applies only when the virtual machine is not running.

**Chassis asset tag:** Sets the chassis asset tag of the specified virtual machine to the value specified (optional). This can be useful for tracking Virtual PCs when using hardware management software or tracking other specific information. Applies only when the virtual machine is not running.

**Window location and size:** Specifies the location and window size of the specified virtual machine (optional). The offsets are relative to the upper left corner of the screen. Applies only when the virtual machine is running.

**Additional command-line options to pass:** Adds the specified options to the Virtual PC command line (optional).

**Override Virtual PC executable filename:** Overrides the default location for the Virtual PC executable (optional). Useful if more than one version of Virtual PC is installed.



### 3.15.4 Virtual Server

The Virtual Server action performs an operation one or more Microsoft Virtual Server virtual machines. See here for options for automating build steps within a virtual machine.

When the step completes, the following temporary macros are created or updated:  
For the Report operation, the state of the last matching PC will be stored in the **VIRTUALSERVER\_VM\_STATE** temporary macro.

#### Operation Tab

#### Parameters Tab

#### Text Tab

*Notes:*

- This action has been tested with Virtual Server 2005 and may work with other versions as well. Either Virtual Server or the Virtual Machine Remote Control Client Plus (including the COM component) must be installed on the machine where Visual Build is installed.
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.15.4.1 Operation Tab

This tab of the Virtual Server action specifies the operation and machine information.

**Operation:** The virtual machine operation to perform.

**Computer:** The name of the computer hosting the virtual machine (optional, blank for local computer).

**Machine:** The name of the virtual machine, or filename when registering a machine (required). For operations other than Register, \* in this field indicates to perform the operation on all virtual machines on the given server, and a regular expression can also be entered in this field to perform the operation on all machine names matching the regular expression.

**Wait for completion:** If checked, the action will wait for the related task to complete before continuing. For the Startup option, if checked, the action will also wait for the guest operating system to start and begin reporting a heartbeat (Virtual Machine Additions must be installed on the guest OS to use this option).

**Log progress:** Causes percent completion of the task to be reported if that information is available (enter blank or 0 to disable).

*Note:* For the Report operation, the state of the last matching PC will be stored in the **VIRTUALSERVER\_VM\_STATE** temporary macro. The state values are as follows:

Invalid	0	
Turned off	1	
Saved	2	
Turning on	3	
Restoring	4	
Running		5
Paused	6	
Saving	7	
Turning off	8	
Merging drives	9	

Delete machine 10

### 3.15.4.2 Parameters Tab

This tab of the Virtual Server action configures parameter names and values for the Set Parameters operation.

**Parameter Name:** Names of parameters to set.

**Parameter Value:** Values of parameters to set.

Enter a parameter name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

### 3.15.4.3 Text Tab

This tab of the Virtual Server action is used to enter the text to send for the *Type text* or *Type key sequence* operations.

For the *Type text* operation, this field must contain ASCII text to simulate being typed in the virtual machine.

For the *Type key sequence* operation, enter a key sequence string (a comma-delimited set of key identifiers) which is used to simulate the key press and release sequence of a standard U.S. 101-key AT-style keyboard. The following text would send the Ctrl+Alt+Del sequence to the virtual machine session:

DOWN, Key\_LeftCtrl, DOWN, Key\_LeftAlt, DOWN, Key\_Delete, UP, Key\_LeftCtrl, UP, Key\_LeftAlt, UP, Key\_Delete

See the Key Identifiers Reference for more details.

## 3.15.5 VirtualBox

This action creates a step to perform an operation on a VirtualBox virtual machine.

**Command:** The virtual machine command to perform.

**Virtual machine:** The name or filename of the virtual machine.

**Additional command-line options:** Adds the specified options to the VirtualBox command line (optional). See the VBoxManage documentation for valid options.

**Override VirtualBox executable filename:** Overrides the default location for the VirtualBox executable (optional).

*Note:* This action has been tested with VirtualBox versions 3 thru 5 and may work with other versions as well.

## 3.15.6 VMware Server

This action creates a step to perform an operation on a VMware Server, GSX Server, or ESX Server virtual machine. See here for options for automating build steps within a virtual machine.

*Notes:*

- The VMware Server 2 / vSphere / Player action can also be used for VMware Server, vSphere, ESX Server, and ESXi (and supports additional operations).
- By default, this action is not displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*

**Operation:** The virtual machine operation to perform.

*Note:* The VmCOM API used by this action does not support all available operations (manipulating snapshots, running programs in the guest OS, etc.). The VMware Workstation action can be used for these operations (for Server v1.0, the VMware Server Programming API client component must be installed for this to be available).

**Computer:** The computer hosting VMware Server (blank for local computer).

**Port:** The port to connect to (blank for default).

**Username:** Username to connect to the server with (optional).

**Password:** Password to use when connecting to the server (optional).

**Machine:** The filename of the virtual machine to operate on. For operations other than Register or Unregister, \* in this field indicates to perform the operation on all registered virtual machines on the given server, and a regular expression can also be entered in this field to perform the operation on all machine filenames matching the regular expression.

**Device:** A device filename for the Connect and Disconnect device operations.

**Wait for completion:** If checked, the action will wait for the related task to complete before continuing. For the Startup option, if checked, the action will also wait for the guest operating system to start and begin reporting a heartbeat (Virtual Machine Additions must be installed on the guest OS to use this option).

**Log every x seconds:** Causes progress to be reported while waiting (enter blank or 0 to disable).

**Variable Name/Value:** Names and values of parameters to set for the Set OS variables and Set configuration variables operations. Enter a variable name and its value in the edit fields, and click Insert to add to the list. Select an item in the list to update its value or delete it from the list.

*Notes:*

- This action has been tested with VMware Server 1.0 and GSX Server 3.2 and may work with other versions as well. The VMware VmCOM component must be installed on the local computer.
- For the Report operation, the state of the last matching PC will be stored in the **VMWARESERVER\_VM\_STATE** temporary macro. The state values are as follows:

On	1
Off	2
Suspended	3
Stuck	4
Unknown	5

### 3.15.7 VMware Workstation / Server 2 / vSphere / Player

This action creates a step to perform a command on a VMware Workstation, Server, vSphere, ESX, ESXi, or Player virtual machine.

**VMware Tab**

**Server Tab**

**Options Tab**

## Advanced Tab

### Remote Tab

#### Notes:

- This action has been tested with VMware Workstation 5 thru 12, Server 1 and 2, vSphere/ESX/ESXi 4 thru 5, Player 3 thru 7, and may work with other versions as well.
- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.15.7.1 VMware Tab

This tab of the VMware Workstation / Server 2 / vSphere / Player action specifies the machine and command properties.

**Command:** The virtual machine command to perform.

<u>Command Type</u>	<u>Minimum Version</u>
Start, stop, reset, suspend, upgrade, install tools	Workstation 5.0, Server 1.0, ESXi 3.5
Snapshots	Workstation 5.5, Server 1.0, ESXi 3.5
Guest OS	Workstation 6.0, Server 2.0, ESXi 3.5
Register, unregister, clone	Server 2.0, ESXi 3.5 (Workstation not supported)
VProbes, recording, replay	Workstation 6.5, ESXi 3.5 (Server not supported)
Pause, unpause, capture screen, variables	Workstation 6.5, Server 2.0, ESXi 3.5

**Virtual machine:** The virtual machine filename to operate on (required for all commands except List virtual machines).

**Host OS username:** Username in guest OS (optional).

**Guest OS password:** Password in guest OS (optional).

**Snapshot, program, script, share, filename, directory, or process ID:** Required for snapshot, recording, replay, variables, capture screen, and guest OS commands.

<u>Command/category</u>	<u>Value</u>
Snapshots	Snapshot name
Run Program	Program to run
Run Script	Interpreter path
File	Filename in guest
Directory	Directory in guest
Share	Share name
Process	Process ID
Begin recording	Snapshot object name
Begin replay	Snapshot object name
Variables	runtimeConfig (.vmx file) or guestEnv (environment variable)
Capture screen	Output path on host
Load VProbes script	Text of VProbes script
Clone	Destination .vmx file path

**Program arguments, script text, host path, destination path, or new filename:** Required for Run program, Run script, File copy, variables, and share commands.

<u>Command/category</u>	<u>Value</u>
Run Program	Program arguments

Run Script	Script text
File	Filename in host
Share	Host path
Variables	Variable name
Clone	Full or linked

**Variable value:** Required for Write variable command.

<u>Command/category</u>	<u>Value</u>
Write variable	Variable value
Clone	Snapshot name (optional)

### 3.15.7.2 Server Tab

This tab of the VMware Workstation / Server 2 / vSphere / Player action specifies properties for a remote VMware host.

**Host:** The remote host where the guest machine is located (optional).

**Host type:** The type of VMware host (optional).

**Port:** The port on the remote host to connect to (optional).

**Username:** Admin login on the server (optional).

**Password:** Admin password on the server (optional).

### 3.15.7.3 Options Tab

This tab of the VMware Workstation / Server 2 / vSphere / Player action configures additional options.

**Prevent prompting:** If checked, the action will enable the `msg.autoAnswer` property in the virtual machine configuration file. *Note:* The `.vmx` or `.vmtm` file must be writeable if checked.

**Don't wait for program to finish:** Returns immediately when running program (applies only to Run Program command).

**Force interactive guest login:** Forces interactive guest login (applies only to Run Program command).

**Ensure window is visible:** Ensures the window is visible (applies only to Run Program command).

**Additional command-line options to pass:** Adds the specified options to the `vmrun` command line (optional).

**Override VMware executable filename:** Overrides the default location for the `vmrun.exe` executable (optional). Useful if more than one version of `vmrun` is installed or if the action is unable to locate the executable automatically. `vmrun.exe` is installed with VMware Workstation and Server, and for other VMware products it can be downloaded here.

## 3.15.8 Windows Virtual PC

The Windows Virtual PC action starts a Windows Virtual PC virtual machine. See here for options for automating build steps within a virtual machine.

**Virtual machine:** The virtual machine filename to start.

**Additional command-line options to pass:** Adds the specified options to the Windows Virtual PC command line (optional).

**Override Windows Virtual PC executable filename:** Overrides the default location for the Windows Virtual PC executable (optional).

## Advanced Tab

### Remote Tab

*Notes:*

- This action has been tested with Windows Virtual PC for Windows 7.
- If Visual Build doesn't detect that this product is installed when first run, this action will not be displayed (existing steps will still be displayed in the Step panes and will build normally). To show all hidden actions, right-click in the Actions pane and choose *Show Hidden*.

#### 3.15.8.1 Virtual PC tab (COPY)

This tab of the Virtual PC action specifies the operation and machine information.

**Operation:** The virtual machine operation to perform.

**Machine name:** The name or filename of the virtual machine.

**Don't start Virtual PC console:** Starts the specified virtual machine without starting Virtual PC Console. This can be useful for simplifying the Virtual PC interface in a corporate or lab environment, where users only want to access a virtual machine and do not need access to Virtual PC Console. Applies only when starting Virtual PC.

**Don't start machines configured to start automatically:** Prevents the starting of any virtual machines that are configured to start automatically. Applies only when starting Virtual PC.

**Turn on host-side disk caching:** Turns on host-side disk caching, which can improve performance of virtual machines running operating systems other than Windows. This can be useful for resolving poor performance problems with disk intensive tasks. Applies only when starting Virtual PC.

**Disable close button:** Disables the Close button on the specified virtual machine. Applies only when starting the virtual machine.

**Clip accelerated S3 bit coordinates to 12 bits:** Clips accelerated S3 bit coordinates to 12 bits on the specified virtual machine. Applies only when starting the virtual machine.

**Network connections operate in external only mode:** Specifies that all network connections on the specified virtual machine operate in external only mode. This limits the access of the virtual machine to only resources external to the physical computer. Applies only when starting the virtual machine.

**Screen mode:** Optionally changes the screen mode between full screen, windowed, and minimized.

**Bring window to foreground:** Brings the specified virtual machine to the foreground of the screen. Applies only when the virtual machine is running.

#### 3.15.8.2 Options tab (COPY)

This tab of the Virtual PC action configures additional options.

**Disable direct execution mode:** Disables any direct mode execution within the specified virtual machine. This can be useful for providing a temporary resolution for poor stability of applications on a guest operating system. Applies only when starting the virtual machine.

**BIOS serial number:** Sets the BIOS serial number of the specified virtual machine to the value specified (optional). This can be useful for tracking Virtual PCs when using hardware management software. Applies only when the virtual machine is not running.

**Chassis asset tag:** Sets the chassis asset tag of the specified virtual machine to the value specified (optional). This can be useful for tracking Virtual PCs when using hardware management software or tracking other specific information. Applies only when the virtual machine is not running.

**Window location and size:** Specifies the location and window size of the specified virtual machine (optional). The offsets are relative to the upper left corner of the screen. Applies only when the virtual machine is running.

**Additional command-line options to pass:** Adds the specified options to the Virtual PC command line (optional).

**Override Virtual PC executable filename:** Overrides the default location for the Virtual PC executable (optional). Useful if more than one version of Virtual PC is installed.

## 3.16 Custom Action Details

Custom actions are used to extend the functionality of Visual Build. Several features are available in all custom actions:

- General properties tab
- Field Overrides
- File Extensions
- Default Properties
- User Defined actions
- Action Registration

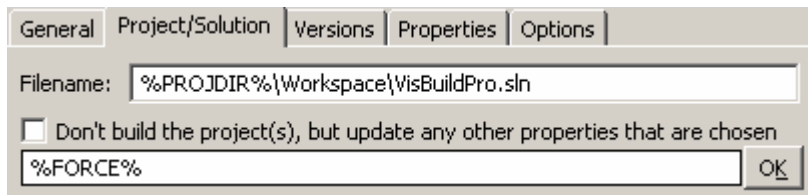
Many pre-defined custom actions are provided, and users can also extend the functionality of Visual Build by creating user-defined custom actions.

### 3.16.1 Field Overrides

Each custom action provides a convenient user interface for entering fields that are pertinent for that action. Visual Build provides great flexibility when entering values for string fields, allowing the use of macros (delimited by percent % chars) and script code (delimited by bracket [ ] chars) within these fields. But sometimes it would be useful to generically define the value for a checkbox, combo box, or radio button field (for instance to have a single step to register or unregister a component or to force or not force a build of a Visual Studio project).

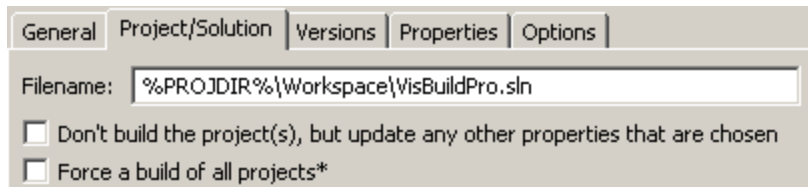
Visual Build provides a field override capability for these situations, allowing any custom action checkbox, radio button, or combo box value to be defined as a string value (which can contain also script or macros) to be evaluated at build time and converted to a boolean (for checkboxes) or integer (for radio buttons and combo boxes) value to determine that field's value. A macro referenced in the override field could be defined by passing it into the build, by prompting the user, etc., dynamically determining what that field's value is.

To enter an override value for one of these fields, tab to the field to override and press F2 or click the Insert Macro button. An override edit control will displayed on top of the checkbox/radio button/combo box, and a value can be entered. The Insert Macro and Script buttons can be used to insert macros or script code, and the field tool tip will displayed just as it is for other string fields.



To undo changes in an override field, type Alt+Backspace, or press Esc to cancel all changes made to the dialog. To accept changes and hide the edit control, click the OK button next to the edit control, tab to another field and press F2 to enter an override for that field, or press Enter to accept all changes made in the dialog.

When an override value has been entered for a field, an asterisk \* will be displayed following that field to indicate that an override exists (press F2 while focus is on the field to edit an existing override). Holding the mouse over the field containing an override will display the override value with any macros expanded. The value of the checkbox, radio button, or combo box will not necessarily correspond to the actual override value.



To remove an existing override, clear out the override edit control value. Or click on the checkbox, combo box, or a radio button in the group. A message box will be displayed confirming that the override value should be removed, and if accepted, the asterisk will be removed from the field and the actual checkbox/radio button/combo box value will be used.

When the step is built, any override fields will be converted to a boolean or integer value (if the value cannot be converted [it is not a valid number], an error will be logged and the build will stop). For checkboxes, a value of 0 evaluates as False and any other number (1, -1, etc) evaluates as True. For radio buttons and drop-down list combo boxes, the value must be a zero-based value corresponding to one of the radio buttons in the group or items in the list.

All checkbox and radio button fields on custom actions screens can be overridden. The only exceptions are the conditional build rule fields on the General tab, which cannot be overridden.

### 3.16.2 File Extensions

Many custom actions are associated with one or more filename extensions. This allows:

- Drag a filename from Explorer and drop onto Visual Build to quickly create a build action for that file.
- Right-click a filename in Explorer and choose *Make/Compile/Run with Visual Build* from the popup menu to create an associated action.
- Create actions by launching the GUI app from the command-line.

The actions with associated extensions are:

Action	Extensions
7-Zip	7z
Advanced Installer	aip
COM Register	dll, ocx, tlb, olb
DeployMaster	deploy
Doc-O-Matic	dox
Doc-To-Help	d2h



Document! X	dxp
Dr.Explain	gui
Flare	flprj
FxCop	fxcop
ExpertInstall / Tarma Installer	tip, tin, im7, im9
Help & Manual	hm2, hm3, hmx, hmxz, hmxp
Helpinator	hgu
HelpMaker	sf?
HelpNDoc	hnd
HelpScribble	hsc
HelpStudio	hsp
HyperText Studio	hts
Inno Setup	iss
InstallAnywhere	iap, iap_xml
InstallAnywhere.NET	iax
InstallAWARE	mpr
Installer2Go	i2g, wip
InstallShield	ipr, ism, pfw, ise, issuite
Make Delphi	bpr, dpr, dpk, bdsproj, bdsgrout, cbproj, dproj, groupproj
Make JBuilder	jpr, jpx, jpgr, bpgr
Make VB6	vbp, vbg, ebp, ebg
Make VC6	dsp, dsw, vcp, vcw
Make VS.NET thru 2012	sln, vcproj, csproj, vbproj, vdproj, icproj, wixproj, lsproj, lsxproj, wdproj, dwproj, rptproj, dtproj, oxygene
MSIStudio	mxp
Mutilizer	mpr, m7p, m7x
NSIS	nsi
RoboHelp	xpj
Run Program	exe, com, bat, cmd, ps1
Sandcastle	shfb
Setup Factory	sf6, sf7, sf8, sufproj, suf
SetupBuilder	sb5, sb6, sb7, sb8, sbp
Sisulizer	slp
Tarma QuickInstall	tin
Tarma InstallMate	im7
Transform XML Log	xml
UNZIP Files	gz, tgz, gzip, tar, taz
Enhanced Unzip Files	zip, zipx, jar, docx, xlsx, pptx, odt, ods, odp
Virtual Server	vmc
VMware Server	vmx
Windows Installer	msi, appx
WiX	wxs
Wise Setup	wse, wsi, wsm
Write INI	ini

### 3.16.3 Default Properties

All custom actions define a default property which is displayed in the Step Panes Default Property column, logged when building if the *Log default property of each step* checkbox is enabled in the logging options dialog, and used to navigate to a filename or folder in the default property.

See the action registration topic for details on customizing the default property for an action.

### 3.16.4 User-Defined Actions

Users can create their own custom actions and plug them into Visual Build's extensible architecture. This can be used to create action components for third party tools not supported by the built-in actions,

to create actions that accept custom inputs for a particular build, encapsulate common functionality used in your builds, and more.

## Action Components

Custom actions have two components:

- Custom tabs that are displayed in the step properties dialog within Visual Build for entering any custom properties used by the action (custom tabs are optional).
- The actual action logic that is performed when the step is built. All information about a user-defined custom action is stored in a `.action` XML configuration file stored in the Action Properties dialog or registered with the COM Register action.

## Creating and Using Actions

A user-defined action is created by activating the Actions pane and choosing *Edit | Insert Action* on the menu bar. This will display the Action Properties dialog to define the action's custom UI pages and the action code itself that is executed when the step is built. Actions are stored and loaded from a `<action name>.action` file in the application data path. After an action has been created, it will be added to the Actions pane and new steps for that action can be created by double-clicking on the action.

## Action Logic

The action logic for user-defined actions can be implemented in one of two ways:

- **Component**

A component action is a COM component created in any development tool supporting creating COM components that implement interfaces, such as Microsoft Visual Studio and Embarcadero Delphi.

The component needs to reference the *Visual Build Professional 9 Server Objects* type library (defined in `System\VisBuildSvr.dll` in the installation path) and implement the *ICustomAction* interface. Action components have access to the Visual Build object model (for running programs, reading and modifying macros and step properties, logging messages, etc.). The return value of the *BuildStep* method determines the success/failure status of the step.

*Note:* See this topic for a discussion of how thread affects actions.

- **Script**

A script action is implemented as script code (in any Active Scripting language).

The action code is implemented in a function named `vblld_BuildStep` (a return value of `vblldStepStatFailed` indicates that the action failed). Action script code has access to the Visual Build object model (for running programs, reading and modifying macros and step properties, logging messages, etc.) and can also access any script functions defined within Visual Build or other COM or .NET components installed on the system.

## Sample Actions

The `User Actions\User Actions.bld` project in the Samples folder (and subfolders) demonstrates custom user actions written in VBScript, VB6, C#, and VB.NET.

In addition, several of the built-in custom actions are implemented as user actions. These actions can be used as templates for your own actions by examining the action's properties, GUI and script code.

## Debugging User Actions

### Components

1. In your user action project's debugging properties, set `C:\Program Files\VisBuildPro9\VisBuildPro.exe` as the debug executable (optionally set the debug executable's arguments to the test .bld filename in double quotes).
2. In your project IDE, set breakpoints in the project as desired, and start debugging the project.
3. In Visual Build, open or create a project and add a step for the user action to the project.
4. Build the project or the user action step.
5. When the step gets built, the breakpoints in your user action will be stopped at, allowing the action to be debugged.

### Script

1. Use the code completing Script Editor to enter the action script code (the Script Editor immediate window can also be used to test functions and code).
2. Create a step for the action and configure its properties.
3. Build the project or step.
4. Any error output will be logged and the build output can be navigated to debug errors. If a syntax or runtime error occurs in the action script code, choose *Go | Last Error* on the menu, and the Script Editor dialog will be opened to the line of code in the script action where the error occurred, allowing the problem to be diagnosed and corrected.

### 3.16.5 Action Registration

Visual Build custom and user actions (their UI and action components) can be registered in two ways:

1) An action's registration can be stored under the registry

`HKEY_LOCAL_MACHINE\Software\Kinook Software\Visual Build Professional 9\Actions`. The COM Register action can be used to register user .action files in this manner. This registration takes precedence over method #2 for actions with the same name. The following manual changes can be made to these registry settings for the built-in actions:

- The `DefaultProperty` value can be modified to change the data displayed in the Step Panes Default Property column (and logged if the *Log default property of each step* checkbox is enabled in the logging options dialog) for a given action. To see the available property names for an action, print or preview the project (ensure that *Detailed project report* is selected and the *Include extended properties* option is checked in the print options dialog) or open a .bld file in a text editor and view the element names of a step for that action.
- The `Category` value can be modified to change the category an action is displayed under in the Actions pane.
- An action's key can be deleted to remove the action from the Actions pane. Custom properties for deleted action steps in existing projects can't be edited, and they will fail to build. Actions can also be hidden from the Actions pane (steps for hidden actions can be built and edited).

2) Any .action files found in the application data path will be loaded on startup. Action files in this directory take precedence over the built-in system actions for action files with the same name.

#### Notes:

- To copy a .action file to the default application data path from Windows Explorer, right-click on the .action file and choose *Install* from the context menu.
- To update the GUI to reflect any registration changes made via the COM Register action or copying/deleting .action files to/in the application data path, select *View | Refresh* on the menu bar (F5).

## 4 Scripting

[Windows Script](#) integration is a powerful feature that can be used throughout your Visual Build projects. Any script language registering itself as an Active Script host engine can be used within Visual Build. [VBScript](#) and [Jscript](#) are provided by Microsoft, but other companies have created script engines for [PerlScript](#), [Python](#), [RubyScript](#), etc. Please contact the vendor of the script engine for details on installing and registering the language for Active Script use.

Script code can be used for complex build rules, accessing common objects such as the Visual Build object model, `FileSystemObject`, [Dictionary](#), `WshShell`, and [MSXML](#) within a project, creating and using custom COM objects that you have created, and more.

Script code can be entered in multiple ways:

1. Script expressions can be entered in any field in a step where the Script Editor button is enabled and in a macro's Value field by enclosing the expression in [ `ScriptExpressionHere` ].  
*Note:* to enter literal bracket characters in a field, use double bracket characters [[ ]].
2. Script code can be executed in a Run Script action.
3. Custom script code can be attached to project and step events.
4. Script functions can be added in the Script Editor, and these functions will be globally available to all script code that is executed.
5. User actions can be implemented in script.

Before a script code fragment, expression, or event is executed, all the script code defined in the Script Editor for that language is added to the engine. The Visual Build Application, Project, Step, and Builder automation objects are also added as named items (the Builder object is only available when building).

Unlike macros, where macros of the same name can override another based on their precedence, all the script code (Temporary, Project, Global, and System) for a language is concatenated together and added as global items that are accessible to all script code.

Script code can also create other COM objects that are available, etc. Several predefined system scripts are provided. See the `Script.bld` sample for ideas on the use of script within projects.

### Notes on Debugging Scripts

- When an error occurs during a build or in the Script Editor Immediate window, as much error information as possible is logged. For well-behaved script engines ([VBScript](#) seems to be best for this), this will include the source location (including event name, script editor tab, and/or step property name if they apply), line and column number, and error description, allowing for easy navigation to the erroneous code.
- The Script Editor Immediate window can be used to evaluate expressions, function calls, etc.
- You can use `Builder.LogMessage` in your script code to trace variable values and debug info when building.

### 4.1 Script Expressions

Script expressions can be inserted into any step field (even radio buttons, checkboxes, and combo boxes via field overrides) or macro value. A script expression is indicated by placing brackets around it (i.e., [ `5+5` ]); the script language used is the default script language set in the application options dialog. When using the Insert button on the Script Editor dialog, the brackets are added automatically.

*Note:* To enter a literal bracket character in a field, type two bracket characters [ [ or ] ].

Script code in a field or macro is evaluated whenever the step is built. Normally, when editing a step, script code is not evaluated for display in a field's tool tip (this allows viewing of the script code after any macros have been expanded). Holding down the Shift key before moving the mouse cursor over the field will cause the script code to be evaluated and displayed in the tool tip.

## 4.2 Script Events

Script events allow custom script code to hook into build events at various points in the build process. This can be useful for performing logic to enable/disable file logging for portions of a build, as an alternative way to implement build rules, to check the step's output and fail the step if the contents indicate an error condition, etc.

Event functions can be created using the Events toolbar button or the Edit menu in the Script Editor. The available events are described below.

### Project Events

Project events are called once for each build. The code for project events should be entered as Project script in the Script Editor (it can also be entered as Global script, but each event function can be defined in only one location).

<u>Event Function</u>	<u>Description</u>
<code>vbld_BuildStarting()</code>	Called before the build starts (before any logging).
<code>vbld_BuildDone(status)</code>	Called after a build completes (after all logging completed); the completion status of the build is passed.
<code>vbld_ProjectLoaded(filename)</code>	Called when a project is loaded; the filename of the project is passed. The <code>_BUILD_PROFILE_</code> temporary macro can be created or updated in this method to define the build profile.

*Note:* Only script code for the default script language will be executed for project events.

### Step Events

#### Step-Specific

Step-specific events are invoked once for each step that is built. The code for these step events must be entered on the Step tab of the Script Editor, which must be opened from the step properties dialog to edit these events. The global Step item is available for accessing the properties of the current step.

<u>Event Function</u>	<u>Description</u>
<code>vbld_StepStarting()</code>	Called after build rules have been evaluated but before the step starts; if False is returned, the step is skipped.
<code>vbld_StepStarted()</code>	Called immediately before the custom action for the step is built.
<code>vbld_StepDone()</code>	Called after the custom action for the step has finished building; the step status is available in the <code>Step.BuildStatus</code> property and can also be updated to change the success/failure status of the step. If the step is marked to continue on failure, throwing or raising an error from this event will not ignore the failure (providing a way to abort the build even when the step is marked to continue on failure). The step's output is available in the <code>LASTSTEP_OUTPUT</code> system macro (accessed from script code via <code>Application.ExpandMacrosAndScript("%LASTSTEP_OUTPUT%")</code> ).

#### Project-Level

Project-level step events are also invoked once for each step that is built, but the code is entered on the Project tab of the Script Editor. The global Step item is available for accessing the properties of the current step.

<u>Event Function</u>	<u>Description</u>
<code>vblD_StepStartingProject()</code>	Called after build rules have been evaluated but before the step starts and before the step-specific <code>vblD_StepStarting</code> event.
<code>vblD_StepStartedProject()</code>	Called immediately before the custom action for the step is executed and before the step-specific <code>vblD_StepStarted</code> event.
<code>vblD_StepDoneProject()</code>	Called after the custom action for the step has finished building and after the step-specific <code>vblD_StepDone</code> event; the step status is available in the <code>Step.BuildStatus</code> property and can also be updated to change the success/failure status of the step. If the step is marked to continue on failure, throwing or raising an error from this event will not ignore the failure (providing a way to abort the build even when the step is marked to continue on failure).

*Notes:*

- Script code for step script events must use the default script language.
- Script event code cannot reference macros directly -- use the script code `Application.ExpandMacrosAndScript("%MYMACRO%")` instead of `"%MYMACRO%"`.
- The `StepStarted` and `StepDone` events do not fire if the step is skipped.

## 4.3 System Scripts

Visual Build several built-in [VBScript](#), [Jscript](#), and [PerlScript](#) system scripts for manipulating files, date and time functions, and more. The scripts can be viewed by opening the Script Editor and changing to the System tab. See the samples for sample usage of many of these scripts.

### Date/Time Functions

### File Utility Functions

### Miscellaneous Functions

### Loop Support Functions

### Obsolete Functions

*Note:* Some functions are defined only for VBScript.

#### 4.3.1 Date/Time Functions

**`vblD_FormatDate()`**: returns the current date in the form YYYYMMDD.

**`vblD_FormatDateD(dte)`**: returns the given date in the form YYYYMMDD.

**`vblD_FormatDateEx(dateStr, format)`**: parses and formats the given date+time as specified.

Available format strings:

<code>yyyy</code>	4-digit year
<code>yy</code>	2-digit year (left padded with 0s)
<code>y</code>	1- or 2-digit year
<code>mmmm</code>	Month name
<code>mmm</code>	Month name (abbreviated)
<code>mm</code>	Month (left padded with 0s)
<code>m</code>	Month
<code>dd</code>	Day (left padded with 0s)
<code>d</code>	Day
<code>hh</code>	24-hr hour (left padded with 0s)

h	24-hr hour
HH	12-hr hour (left padded with 0s)
H	12-hr hour
MM	Minutes (left padded with 0s)
M	Minutes
SS	Seconds (left padded with 0s)
S	Seconds
AP	AM/PM
A	A/P
ap	am/pm
a	a/p

**vblid\_FormatDateTime():** returns the current date+time in the locale-neutral form DYYYYMMDDTHHMMSS, useful for storing a timestamp in a macro.

**vblid\_FormatDateTime(dte):** returns the given date+time the locale-neutral form DYYYYMMDDTHHMMSS.

**vblid\_FormatTime():** returns the current time in the form HHMMSS.

**vblid\_FormatTimeT(dte):** returns the given time in the form HHMMSS.

**vblid\_ParseFormattedDateTime(dateStr):** returns a string formatted with vblid\_FormatDateTime as a date variable, useful for retrieving from a macro.

### 4.3.2 File Utility Functions

**vblid\_FSO():** creates and returns a new FileSystemObject for manipulating and retrieving properties of files and folders. Some common uses for this object:

<u>Script expression</u>	<u>Purpose</u>
[vblid_FSO.FolderExists("%SAMPLEDB%")]	Determine if a folder exists (returns -1 if found or 0 if not)
[vblid_FSO.FileExists("c:\autoexec.bat")]	Determine if a file exists (returns -1 if found or 0 if not)
[vblid_FSO.GetFileVersion("c:\pathto\file.ext")]	Returns the file version of the specified file

For more information, see the documentation for the [FileSystemObject](#).

**vblid\_CompareFileDates(TargetFile, SourceFile):** Provides a simple 'make' capability to compare a source and target file. Compares the date/timestamp of two files and return -1 if the target file does not exist or is older than the source file, 0 if they are equal, or 1 if the target file is newer than the source file.

**vblid\_CopyFile(SourceFile, DestFile, Overwrite):** Copies a file. *Note:* The Copy Files action provides more robust file copy capabilities.

**vblid\_FileDateModified(FileSpec):** Returns the modification date of a file.

**vblid\_FileOutOfDate(TargetFile, SourceFile):** Compares files by version info if available or size+timestamp if not if the target file does not exist, returns True; otherwise if the files contain version information, the version info is compared and True is returned if the target file's version is older; if no version info is available, the files are compared by timestamp of last modification, returning True if the target file is older; if the timestamps match, a file size comparison is also performed and True returned if target file is older than the source file.

**vbld\_GetFileContents(Filename):** Retrieve the contents of a file and double up characters that are treated specially by Visual Build. Bracket characters [ ] and percent sign % characters are normally interpreted by Visual Build as referencing script code and macros within a field. When retrieving the contents of a file into a field, using this function will cause these characters to be treated as literals by Visual Build. *Note:* The Read File action can also retrieve the contents of a file.

**vbld\_MakeFileWriteable(Filename):** ensure that a file is writeable and return True if found and succeeded. *Note:* The Set File Attributes action can also be used to make a file writeable (or set other attributes).

### 4.3.3 Miscellaneous Functions

**vbld\_AllMacros():** return the collection of all defined macros, including each macro with highest precedence for each macro with the same name defined for multiple macro types.

**vbld\_AppIsRunning(title, closeIt):** Determine if an app with the given window title is running, and attempt to close if closeIt = True.

**vbld\_FindProcess(name):** Look for processes with the given name (i.e., notepad.exe) and return their process IDs (one per line).

**vbld\_EscapeString(str):** Double up characters that are treated specially by Visual Build. Bracket characters [ ] and percent sign % characters are normally interpreted by Visual Build as referencing script code and macros within a field. When a string or macro value will be used in a field, call this function from script code on the value to cause these special characters in a string to be treated as literals by Visual Build.

**vbld\_If(expression, truePart, falsePart):** Returns one of two objects/values, depending on expression.

**vbld\_MSXML():** creates and returns a new MSXML parser object. MSXML v3 or later must be installed.

**vbld\_PadLeft(str, length, padChar):** Pads a string on the left to the specified length with the specified pad character, returning the padded string.

**vbld\_QuoteStr(str):** Properly quotes a string for passing as a command-line argument.

**vbld\_StepProp(propName, varType):** Retrieves a step property value with all macros and script expanded and converted to the specified type (i.e., vbString/VT\_BSTR [8], vbLong/VT\_I4 [3], vbBoolean/VT\_BOOL [11]).

**vbld\_TempMacros():** return the collection of temporary macros.

**vbld\_TempMacro(macroName):** return the specified temporary macro.

**vbld\_TempMacroObj(macroName):** return the object stored in the specified temporary macro.

### 4.3.4 Loop Support Functions

The following functions are useful when one or more steps need to be repeated once for each line of a file, record in a database, etc., or to implement a stack construct within a build.

*Note:* In Visual Build v7 and later, the Loop action should be used instead of these functions.

**vbld\_AddDelimValue(macroName, value):** Creates a new temporary macro with the value or adds a string to the end of the given temporary macro, separating each value with a Tab character delimiter, and returns the updated value.



**vbld\_NextDelimValue(macroName):** Given the name of a macro containing Tab-delimited strings (populated via `vbld_AddDelimValue`), removes the **first** delimited string from the macro and returns or returns Null if the macro does not exist.

**vbld\_PushDelimValue(macroName, value):** Creates a new temporary macro with the value or adds a string to the end of the given temporary macro, separating each value with a Tab character delimiter, and returns the updated value.

**vbld\_PopDelimValue(macroName):** Given the name of a macro containing Tab-delimited strings (populated via `vbld_AddDelimValue`), removes the **last** delimited string from the macro and returns or returns Null if the macro does not exist.

### 4.3.5 Obsolete Functions

The following system script functions are no longer included with Visual Build Pro. Either convert your projects to use the listed replacement actions, or copy the script code below into your global script code.

*Note:* the Run Program action can be used in place of this script and provides a more robust way to remotely execute programs.

**vbld\_RemoteExecuteCmd(strServer, strCommand, strUserName, strPassword):** Execute a command on a remote computer and return the exit code.

*Note:* the Service action can be used in place of these scripts and provides a simpler method for stopping and starting services.

**vbld\_ConnectServer(strServer, strUserName, strPassword):** Connect to a remote server using WMI and return the server object.

**vbld\_GetService(strServer, strServiceName, strUserName, strPassword):** Retrieve a WMI object reference to a remote service.

**vbld\_RemoteStartService(strServer, strServiceName, strUserName, strPassword):** Start a remote service, returning True if started or False if already running.

**vbld\_RemoteStopService(strServer, strServiceName, strUserName, strPassword):** Stop a remote service, returning True if stopped or False if already stopped.

*Note:* the COM+ Application and COM+ Component actions provide more functionality and should be used instead of the following scripts.

**vbld\_CreateApplication(strName, blnServer):** Create/update a COM+ application, returning True if it did not exist or False if it already existed.

**vbld\_DeleteApplication(strName):** Delete a COM+ application, returning True if it existed or False if it did not exist.

**vbld\_AddComponent(strApp, strDll):** Install a component into a COM+ application, returning True if added or False if the application was not found.

**vbld\_ExportApplication(computer, appName, filename, options):** Export a COM+ application or proxy to a file (see the script code for a description of the parameters).

**vbld\_InstallApplication(computer, filename, destDir, options, user, pwd, RSN):** Install a COM+ application from a file (see the script code for a description of the parameters).

*Note:* the IIS and IIS Virtual Dir actions provides more functionality and should be used instead of the following scripts.

**vbld\_CreateVirtualDir(strPath, strName):** Create/update an IIS virtual directory on the default web site to the specified path.

**vbld\_DeleteVirtualDir(strName):** Delete an IIS virtual directory on the default web site.

**vbld\_CreateQueue(strName, strLabel):** Create the specified MSMQ queue if it doesn't exist, returns True if did not exist and successfully created or False if it already exists.

**vbld\_DeleteQueue(strName):** Delete the specified MSMQ queue if it exists, return True if deleted or False if it doesn't exist.

**vbld\_CreateShare(path, shareName):** Create a network disk share of the given name to the path specified on the local computer.

**vbld\_DeleteShare(shareName):** Delete the given network disk share from the local computer if it exists.

## Obsolete Script Code

```
' Connect to a remote server using WMI and return the server object
Function vbld_ConnectServer(strServer, strUserName, strPassword)

    Dim objLocator

    'Create Locator object to connect to remote CIM object manager
    Set objLocator = CreateObject("WbemScripting.SWbemLocator")

    'Connect to the namespace which is either local or remote
    Set vbld_ConnectServer = objLocator.ConnectServer(strServer,
"root\cimv2", strUserName, strPassword)
    vbld_ConnectServer.Security_.impersonationlevel = 3

End Function

' Retrieve a WMI object reference to a remote service
Function vbld_GetService(strServer, strServiceName, strUserName,
strPassword)

    Dim objServer

    Set objServer = vbld_ConnectServer(strServer, strUserName,
strPassword)
    Set vbld_GetService = objServer.Get("Win32_Service=" & strServiceName
& "'")

End Function

' start a remote service, returning True if started
' use "", "" for username and password if the current user has remote
```

```
execute rights,
' otherwise specify a valid username/password for the remote server
Function vbld_RemoteStartService(strServer, strServiceName, strUserName,
strPassword)

    Dim objService

    Set objService = vbld_GetService(strServer, strServiceName,
strUserName, strPassword)
    vbld_RemoteStartService = (objService.StartService() = 0)

End Function

' stop a remote service, returning True if stopped
' use "", "" for username and password if the current user has remote
execute rights,
' otherwise specify a valid username/password for the remote server
Function vbld_RemoteStopService(strServer, strServiceName, strUserName,
strPassword)

    Dim objService

    Set objService = vbld_GetService(strServer, strServiceName,
strUserName, strPassword)
    vbld_RemoteStopService = (objService.StopService() = 0)

End Function

' create/update a COM+ application, returning True
' if it did not exist
Function vbld_CreateApplication(strName, blnServer)

    Const COMAdminActivationInproc = 0
    Const COMAdminActivationLocal = 1

    Dim cllApps, objApp, i

    vbld_CreateApplication = False
    Set cllApps = vbld_ApplicationCollection
    i = vbld_FindCollectionIndex(cllApps, strName)
    If i >= 0 Then ' update existing item if found
        Set objApp = cllApps.Item(i)
    Else ' add new app if not found
        Set objApp = cllApps.Add
        objApp.Value("Name") = strName
        vbld_CreateApplication = True
    End If

    If blnServer Then
        objApp.Value("Activation") = COMAdminActivationLocal
    Else
        objApp.Value("Activation") = COMAdminActivationInproc
    End If
    cllApps.SaveChanges

End Function

' delete a COM+ application, returning True
' if it existed
Function vbld_DeleteApplication(strName)
```

```

    Dim objCat, cllApps, objApp, i

    vbld_DeleteApplication = False

    Set objCat = CreateObject("COMAdmin.COMAdminCatalog")
    Set cllApps = objCat.GetCollection("Applications")

    i = vbld_FindCollectionIndex(cllApps, strName)

    If i >= 0 Then
        objCat.ShutDownApplication(strName)
        cllApps.Remove i
        cllApps.SaveChanges
        vbld_DeleteApplication = True
    End If

End Function

' add a component to a COM+ application, returning True
' if added or False if the application was not found
Function vbld_AddComponent(strApp, strDll)

    Dim objCat, cllApps, objApp, i

    vbld_AddComponent = False
    Set objCat = CreateObject("COMAdmin.COMAdminCatalog")
    Set cllApps = objCat.GetCollection("Applications")

    i = vbld_FindCollectionIndex(cllApps, strApp)

    If i >= 0 Then
        objCat.InstallComponent cllApps.Item(i).Key, strDll, "", ""
        vbld_AddComponent = True
    End If

End Function

' locate the given COM+ admin collection object by name and return
' its 0-based index or -1 if not found
Function vbld_FindCollectionIndex(cll, strName)

    Dim i, obj

    cll.Populate

    vbld_FindCollectionIndex = -1
    i = 0
    For Each obj In cll
        If LCase(obj.Name) = LCase(strName) Then
            vbld_FindCollectionIndex = i
            Exit For
        End If
        i = i + 1
    Next

End Function

' create an return the COM+ Applications collection
Function vbld_ApplicationCollection()

```

```
Dim objCat, cllApps

Set objCat = CreateObject("COMAdmin.COMAdminCatalog")
Set cllApps = objCat.GetCollection("Applications")
cllApps.Populate
Set vbld_ApplicationCollection = cllApps

End Function

// Create the specified MSMQ queue if it doesn't exist,
// Returns True if did not exist and successfully created
function vbld_CreateQueue(strName, strLabel)
{
    var MQ_PEEK_ACCESS = 32;
    var MQ_DENY_NONE = 0;

    // create the queue info object
    var objQInfo = new ActiveXObject("MSMQ.MSMQQueueInfo");
    objQInfo.PathName = strName;

    var objQ = null;
    // will fail if it doesn't exist
    try
    {
        objQ = objQInfo.Open(MQ_PEEK_ACCESS, MQ_DENY_NONE); //
attempt to open the queue
    }
    catch (E)
    {}

    // if queue not found, create it
    if (objQ == null)
    {
        objQInfo.Label = strLabel;
        objQInfo.Create(false, false); // non-transactional,
non-world-readable
        return true;
    }
    return false;
}

// Delete the specified MSMQ queue if it exists, return True if deleted
function vbld_DeleteQueue(strName)
{
    var MQ_PEEK_ACCESS = 32;
    var MQ_DENY_NONE = 0;

    // attempt to create queue
    var objQInfo = new ActiveXObject("MSMQ.MSMQQueueInfo");
    objQInfo.PathName = strName;

    var objQ = null;
    // will fail if it doesn't exist
    try
    {
        objQ = objQInfo.Open(MQ_PEEK_ACCESS, MQ_DENY_NONE); //
attempt to open the queue
    }
    catch (E)
```

```

    {}

    // if queue not found, create it
    if (objQ != null)
    {
        objQInfo.Delete();
        return true;
    }
    return false;
}

' COMAdminApplicationExportOptions enum values
Const COMAdminExportNoUsers = 0
Const COMAdminExportUsers = 1
Const COMAdminExportApplicationProxy = 2
Const COMAdminExportForceOverwriteOfFiles = 4
Const COMAdminExportInI0Format = 16

' export a COM+ application or proxy to a file
' computer: computer to export from or "" for local computer
' appName: name of application to export
' filename: file to export the application to or "" to export to default
directory
' options: sum of one or more values from COMAdminApplicationExportOptions
(above)
Sub vbld_ExportApplication(computer, appName, filename, options)

    Dim objCat

    Set objCat = CreateObject("COMAdmin.COMAdminCatalog")
    objCat.Connect computer
    objCat.ExportApplication appName, filename, options

End Sub

' create/update an IIS virtual directory to the specified
' path
Sub vbld_CreateVirtualDir(strPath, strName)

    Dim objSvr, objDir

    Set objSvr = GetObject("IIS://LocalHost/w3svc/1/Root")

    On Error Resume Next      ' delete virtual directory if it exists
    objSvr.Delete "IISWebVirtualDir", strName
    Err.Clear

    Set objDir = objSvr.Create("IISWebVirtualDir", strName)

    ' set the permissions to read and call script
    objDir.AccessRead = True
    objDir.AccessScript = True
    ' set path to passed in value
    objDir.Put "Path", strPath
    objDir.SetInfo
    objDir.AppCreate True      ' True = InProc
    objSvr.SetInfo

End Sub

```

```
Sub vbld_DeleteVirtualDir(strName)

    Dim objSvr

    Set objSvr = GetObject("IIS://LocalHost/w3svc/1/Root")

    On Error Resume Next      ' delete virtual directory if it exists
    objSvr.Delete "IISWebVirtualDir", strName
    Err.Clear

End Sub

' execute a command on a remote computer and return the result
' use "", "" for username and password if the current user has remote
execute rights,
' otherwise specify a valid username/password for the remote server
Function vbld_RemoteExecuteCmd(strServer, strCommand, strUserName,
strPassword)

    Dim objInstance, objServer
    Dim intProcessId, intStatus

    'Establish a connection with the server.
    Set objServer = vbld_ConnectServer(strServer, strUserName,
strPassword)

    ' retrieve a process handle
    Set objInstance = objServer.Get("Win32_Process")

    ' run the command
    vbld_RemoteExecuteCmd = objInstance.Create(strCommand, Null, Null,
intProcessId)

End Function

' create a disk share on the local computer with the given path and name
Sub vbld_CreateShare(sharePath, shareName)

    Dim objServer, objShare

    'Establish a connection with the server.
    Set objServer = vbld_ConnectServer("", "", "")

    ' retrieve a process handle
    Set objShare = objServer.Get("Win32_Share")
    objShare.Create sharePath, shareName, 0, Null, "", Null, Null

End Sub

' delete the given network disk share from the local computer
Sub vbld_DeleteShare(shareName)

    Dim objServer, objShare

    'Establish a connection with the server.
    Set objServer = vbld_ConnectServer("", "", "")

    ' retrieve a process handle
    Set objShare = Nothing
    On Error Resume Next
```

```

Set objShare = objServer.Get("Win32_Share='" & shareName & "'")
On Error Goto 0
If Not objShare Is Nothing Then objShare.Delete()

```

End Sub

## 5 Samples

Several sample projects are included to demonstrate some of the ways that Visual Build can be used. Use the samples as a pattern for creating your own automated builds. See the Visual Build [FAQ](#) for additional samples and answers to common questions.

The samples are installed in the *Samples* subdirectory within the Visual Build installation path (accessible from the Windows Start Menu under *Visual Build Professional 9 -> Samples*). The details for each step can be viewed by displaying the step properties dialog for the step. For fields containing macros, hold the mouse over that field to view the expanded value. Viewing the project macros can also be instructive. When building a project, the build output is displayed in the lower Output pane.

*Note:* Some of the samples require administrator rights (Visual Build must be explicitly run as administrator on Windows Vista/2008/7) in order to build successfully.

<u>Sample</u>	<u>Actions Demonstrated</u>
Advanced.bld	Log Message, Run Program, Run Script, Set Macro, Subroutine Call, Wait, Write File, Loop
Chain.bld	VisBuildPro Project
Embarcadero.bld	Make Delphi / C++Builder / RAD Studio
Files.bld	Burn CD/DVD, Copy Files, FTP, Rename Files, Replace in File, Write INI, Set Macro, Write File, Loop, Read INI
Logging.bld	Set Macro, Write File, Log Message, Transform XML Log
Network.bld	FTP, HTTP, Newsgroup Post, Telnet, Send Mail
Prompt.bld	Log Message, Run Script, Set Macro
Recurse.bld	FTP, Process Files, Run Program, ZIP Files, UNZIP Files
RegEdit.bld	Log Message, Set Macro, Write File, Write Registry, Read Registry
Script.bld	Process Files, Run Script
SingleInstance.bld	Delete Files, Exit, Write File
Server.bld	ADO, COM+ Application, COM+ Component, Copy Files, Run Script, Run SQL, Service, Write File
XML.bld	Log Message, Run Script, Loop, Read XML, Write XML
<b>Version Control folder</b>	
ClearCase.bld	ClearCase
ContinuousIntegration.bld	Subroutine Call, Wait, VisBuildPro Project, SourceSafe, Perforce, Set Macro, Run Script, Loop
CVS.bld	CVS
Perforce.bld	Perforce, Copy Files
PVCS.bld	PVCS, Write File, Create Folder
StarTeam.bld	Run Program
Subversion.bld	Subversion
SurroundSCM.bld	SurroundSCM
Vault.bld	Vault
<b>Visual Studio folder</b>	
GetProjVer.bld	Get VS.NET Version, Get VS6 Version
SourceSafe.bld	Copy Files, SourceSafe
Team System.bld	Team Foundation, Team Build, Team Test
TDD.bld	NAnt, NDoc, NUnit
VStudio.bld	COM Register, Make VC6, Make VB6, Make VS.NET, Send Mail,



## SourceSafe

**Miscellaneous folder**

SaveStatus.bld	Log Message, Run Script
TestVFP.bld	Run Script
WebLauncher	Launching a build from a web page front-end

Some sample projects are also provided to demonstrate advanced uses of Visual Build:

- User-Defined Action Samples
- ObjectModel Samples
- VBPLogger Sample
- VisBuildPro samples

## 5.1 Advanced.bld Sample

This sample demonstrates how to:

- 1) Implement if/then logic using conditional build rules containing script.
- 2) Write the contents of a file to the build log.
- 3) Call a subroutine.
- 4) Set up a continuous build process using a repeating build rule.
- 5) Process a set of steps once for each value in a list stored in a macro.
- 6) Update environment variables via the Set Macro action so that the updated variables can be accessed by external programs.
- 7) Use of nested steps to build up relative paths for Run Program step Start In paths.

## 5.2 Chain.bld Sample

This sample shows how to call another Visual Build project, passing different temporary macro values. It calls the RegEdit.bld sample to create/update registry values. Build the project, and view the registry values. Then check the second step and build the project, and the values will change. Build the third step and the values will be deleted.

When built in the GUI App and single-stepped (F10), the chained projects will also be single-stepped for easy debugging.

If the project is built with the Console app:

```
"C:\Program Files\VisBuildPro9\VisBuildCmd.exe" "C:\Program Files\VisBuildPro9\Samples\Chain.bld"
```

The chained project will also be launched in another Console app instance.

The next set of steps demonstrate using the Visual Build action with a Process Files action to build all .bld files except the current one.

The last section shows how to set up parallel builds for performing multiple build operations (which don't depend on each other) simultaneously, then continuing with the main build after all parallel portions have completed:

1. The master build launches two child builds from a VisBuildPro Project step marked to not wait for completion.
2. The master build contains a Wait step so that the rest of the build will not continue until all parallel builds have completed.

## 5.3 Embarcadero.bld Sample

This sample demonstrates using the Make Delphi action to build Embarcadero/CodeGear Delphi and C++Builder projects.

## 5.4 Files.bld Sample

This sample demonstrates using:

- 1) The Burn CD action to backup to DVD
- 2) The Copy Files action to perform an incremental backup
- 3) An incremental web site update using the FTP action
- 4) The Rename Files action to rename .ini files
- 5) The Replace in File action
- 6) Reading and writing INI files
- 7) Performing a set of steps once for each line in a file
- 8) Updating VC6 and VC7 Include path settings

## 5.5 Logging.bld Sample

Demonstrates the use of script events to delete an XML log file at the start of a master project and to disable and re-enable file logging during a build, using project-level file logging, sending an XML log file (requires closing tags to be added), converting an XML log file to an HTML log report, and showing the log file. The XML steps will only be performed if logging format option is set to *XML*.

The LOGFILE project macro overrides the application-wide settings to create a project-specific log file.

*Notes:*

- To make the build log file accessible to other users, simply map an IIS or other web server virtual directory to the path containing the log or HTML files, and the log files can be accessed via any web browser.
- It is recommended that any active scanning anti-virus software be disabled on the build box, as this can interfere with Visual Build writing to its log file (and also slows down builds).

## 5.6 Network.bld Sample

This sample demonstrates the networking actions available in Visual Build:

- FTP
- Telnet
- Newsgroup Post
- HTTP
- Send Mail

The built-in network actions support [SSL/TLS/PCT](#) and SSH/SFTP for secure authentication and data transfer.

## 5.7 Prompt.bld Sample

This sample demonstrates several methods to prompt the user for a value during a build. The simplest way is to reference a non-existent macro and let Visual Build prompt for the value when first referenced during a build (*note*: this applies only to the GUI app and not the Console app). Another method is to create a Run Script step which prompts for the value and stores it in a temporary macro.

To provide even greater control over prompted values (for instance, to provide custom drop-down choices or if you have several values you wish to prompt for), create a custom user action which

prompts for all values and stores the user inputs as macros, or create a custom front-end to launch Visual Build, passing in the user inputs. Custom VB6 and C# front-ends are demonstrated in the ObjectModel samples and the VisBuildPro BuildLauncher sample.

## 5.8 Recurse.bld Sample

This sample demonstrates use of the Process Files action for performing operations on multiple files in a single directory and recursive searching of subdirectories for matching files, the Enhanced Zip Files and Enhanced Unzip Files action for adding or extracting from ZIP files, and more. It includes samples for uploading files to an FTP server, compiling Java source files, building VB and VC++ projects, and deleting files.

## 5.9 RegEdit.bld Sample

This sample shows how to use the Write Registry and Read Registry actions to create, update, and delete registry values. It also demonstrates the use of conditional build rules. The first set of steps create two registry values only if the macro DELETE is not defined; the second set of steps deletes the registry values if the macro DELETE is defined. The macro is not defined for this project, so the values will be created. Build the project, and then open the registry editor (by running RegEdit.exe and view the values that were created under HKEY\_CURRENT\_USER\Software\Kinook Software\Visual Build Professional 9\Test Key).

Use the Chain.bld sample to see how to call this project so that DELETE is defined. Or, create a temporary or global macro called DELETE; then rebuild the project and the values will be deleted.

## 5.10 Script.bld Sample

This sample demonstrates how to integrate and call custom scripts for complex build capabilities.

1. Dictionary: Demonstrates initializing a collection of values using the [Dictionary](#) object, storing in a temporary macro, and using in a later step.
2. Numeric comparison: Using a VBScript expression to perform a numeric comparison and conditionally build the step.
3. Increment macro: How to call a script to increment a build number macro and then use the macro in the following step.
4. Process files: The next set of steps uses the Process Files action and a Run Script step to perform an incremental copy of the sample files to another directory. The first time the project is built, all the files are copied, but if built again, it detects that all files are up-to-date and does not copy them. *Note:* The Copy Files action could also be used to do this, but this technique could be useful if other processing needed to be performed if a file was out of date.
5. Miscellaneous: How to create a shortcut from script, check for an application running, and using OLE Automation to modify an Excel spreadsheet.
6. Object model: Using the Visual Build object model to:
  - Log all steps and properties in a project
  - Dynamically create a new project and build it. This could be useful if a build is driven by the contents of an external configuration file that determines what to build. Steps and macros can be added to the project, their properties set, and the project built to build all steps. First, create a template step manually inside of Visual Build, then open the .bld file in a text editor and view that step's child elements to see what properties need to be set (or copy and paste the *Log all steps and properties* step from this sample into your project and build to see all steps and properties). The action attribute is passed into the step Add method.
  - Show and manipulate the build objects during a build, and then logging the changes that were made. This can be useful to configure Visual Build on a clean build box with the values pulled from config and macro files stored in source control, for instance.

The Logging.bld sample demonstrates some additional scripting techniques.

## 5.11 SingleInstance.bld Sample

This sample demonstrates a method of allowing only a single instance of a build at a time. At the start of the build, it checks for the existence of a marker file, and aborts the build if the marker file is found (and is not more than 5 hours old). Otherwise, it creates the marker file, performs the build, and deletes the marker file when the build completes.

## 5.12 Server.bld Sample

Sample demonstrating:

- Configuring a COM+ application
- Initializing an IIS virtual directory and deploying files
- Stopping and starting services (local and remote)
- Building a project on a remote server
- Performing database operations using OSQL and ADO

The remote steps require that Visual Build be installed on multiple computers, and the appropriate security permissions configured. The samples download and use the [PsExec](#) utility from [SysInternals](#) if not available in the System directory.

The SQL Server steps demonstrate how Visual Build can be used to call a script to update a SQL Server database, but the script it calls doesn't exist and needs to be created and the step updated to point to a valid database. SQL scripts can be used to generate the tables, stored procedures, etc., and to populate tables with data. See the SQL Server Books Online for more details on using OSQL.

This ADO steps demonstrate executing SQL statements and queries on an Access database using ADO. It creates a table to a sample database, adds rows, queries the table's data, and uses the query results in later build steps. Similar steps could also be used for connecting to various other types of databases and servers (Active Directory, ODBC, Oracle, Exchange, SQL Server, etc.).

## 5.13 XML.bld Sample

This sample demonstrates reading and writing XML documents from a build.

## 5.14 Version Control

### 5.14.1 ClearCase.bld Sample

Demonstrates integrating the ClearCase source control product into the build process.

A demo VOB and two demo views are created (named VBP\_DEMO\_VOB, VBP\_DEMO\_VIEW and VBP\_DEMO\_BUILD\_VIEW respectively) and a small Visual Studio solution is loaded into the VOB using the first view. Using the 2nd "build" view, some typical build steps are demonstrated (checkout, checkin, label, update, etc). Finally the demo VOB and views are removed.

*Note:* Since ClearCase/ClearCase LT are client/server systems, a completely isolated server is not used and demo data will be loaded onto your ClearCase server when this build is executed. While using a demo VOB should eliminate any conflict with your existing data, please use caution when executing this demo.

*Note:* This sample project is located in the `Version Control` subfolder of the `Samples` path.

### 5.14.2 ContinuousIntegration.bld Sample

This sample demonstrates performing continuous integration builds with Visual Build. When built, the project runs continuously, building every 10 minutes (configured in the LOOP\_DELAY project macro). For each iteration, the project checks for any changes in the source control repository since the last

build (stored in the LAST\_BUILD\_TIME global macro) and performs the build logic if changes have been made.

The sample demonstrates using Perforce, SourceSafe, Subversion, Surround SCM, Team Foundation, or Vault / Fortress as the source control repository to monitor (check the appropriate Check step for your system). Each check step uses a vblld\_StepDone script event to parse the output and determine if changes have been made since the last build. To incorporate into your builds, change the *Perform Build* step to call your build project instead of the VStudio.bld sample. To use with other SCM systems, create a custom step for your system modeled after an existing check step.

One way to start this build on a dedicated build machine is to create a scheduled task that is scheduled at System Startup. *Note:* No window is displayed for the task when configured in this manner. If this is desired (for instance to monitor via Remote Desktop), Windows can be configured to login automatically at startup (<http://www.google.com/search?hl=en&lr=&ie=UTF-8&q=auto+login+windows+nt>) and the task scheduled to run at Logon.

*Notes:*

- This sample project is located in the Version Control subfolder of the Samples path.
- The project requires VBScript as the default script language.
- You can also perform continuous integration by invoking the Visual Build command-line interface from CruiseControl .NET's Executable task.

### 5.14.3 CVS.bld Sample

The sample shows how to use the CVS action to integrate the CVS version control system into the build process. It shows several different commands being performed on a repository, including checkout, commit, diff, init, import, log, and rtag. The CVS repository is created at %TEMP%\VisBuildPro\CVS\repository.

This sample requires the CVS client app (cvs.exe) to be in the PATH environment variable. CVS can be downloaded from <http://ccvs.cvshome.org/servlets/ProjectDownloadList>, and a Windows binary of CVS 1.11.1 is also available from <ftp://ftp.kinook.com/cvs1.11.1.zip>.

*Note:* This sample project is located in the Version Control subfolder of the Samples path.

### 5.14.4 Perforce.bld Sample

Demonstrates how to integrate the Perforce version control product into the build process.

*Note:* This sample project is located in the Version Control subfolder of the Samples path.

### 5.14.5 PVCS.bld Sample

Demonstrates how to integrate the PVCS Version Manager source control product into the build process.

*Note:* This sample project is located in the Version Control subfolder of the Samples path.

### 5.14.6 StarTeam.bld Sample

Demonstrates integrating the Borland StarTeam source control product into the build process.

*Note:* This sample project is located in the Version Control subfolder of the Samples path.

### 5.14.7 Subversion.bld Sample

This sample shows how to use the Subversion action to integrate the [Subversion](#) version control system into the build process. It shows several different subcommands being performed on a repository. The Subversion repository is created at `%TEMP%\VisBuildPro\Subversion\repository`.

This sample requires the Subversion (`svn.exe`) to be in the PATH environment variable (the Subversion Windows installer updates this variable with the install path).

*Note:* This sample project is located in the `Version Control` subfolder of the `Samples` path.

### 5.14.8 Surround SCM.bld Sample

This sample shows how to use the Surround SCM action to integrate the [Subversion](#) version control system into the build process. It shows several different commands being performed on a repository.

*Note:* This sample project is located in the `Version Control` subfolder of the `Samples` path.

### 5.14.9 Vault.bld Sample

This sample shows how to use the Vault / Fortress action to integrate the SourceGear [Vault](#) / Fortress version control system into the build process. It shows several different commands being performed on a repository.

*Note:* This sample project is located in the `Version Control` subfolder of the `Samples` path.

## 5.15 Visual Studio

### 5.15.1 GetProjVer.bld Sample

Demonstrates retrieving and displaying the file version of Visual Studio 6.0 and .NET projects.

*Note:* This sample project is located in the `Visual Studio` subfolder of the `Samples` path.

### 5.15.2 SourceSafe.bld Sample

This sample demonstrates various ways the SourceSafe action can be used. It creates a sample SourceSafe project, which can be browsed by starting the SourceSafe Explorer, choosing File|Open, navigating to `<TEMP_path>\VisBuildPro\VStudio\Database`, clicking OK, and entering a user of Guest (blank password).

The project demonstrates many of the available SourceSafe database operations, such as Add, Checkin, Checkout, CP, Create, Diff, Directory, Get, History, Label, Paths, Properties, and View.

*Note:* This sample project is located in the `Visual Studio` subfolder of the `Samples` path.

### 5.15.3 TDD.bld Sample

This sample demonstrates the actions for incorporating [NUnit](#), [NDoc](#), and [NAnt](#) into Visual Build automated builds. The `.action` files implementing the custom actions are installed in the application data path and provide actions for the following .NET TDD tools:

- NUnit

- NDoc
- NAnt

The steps of this project build sample steps for each action.

*Note:* This sample project is located in the `Visual Studio` subfolder of the `Samples` path.

#### 5.15.4 Team System.bld Sample

Demonstrates integrating the Team Foundation, Team Test, and Team Build into the build process.

A demo workspace is created (named: `ComputerName + '_DEMO'`), and a small Visual Studio solution (including a simple C# application and test project) is added to your Team Foundation repository via this workspace. Typical build steps are demonstrated (add, checkout, checkin, label, status, etc). Finally, the demo files and workspace are deleted.

*Note:* A completely independent Team Foundation repository is not created in this demo, meaning that demo data will be loaded into your live Team Foundation server. While the use of a demo workspace (which is defined to use a temp path in a temp folder) should eliminate any conflict with your existing data, please use caution when executing this demo. Also be advised that once files are added to a Team Foundation repository, they are "always there" (they are never truly deleted). Please keep this in mind when using this demo.

*Note:* This sample project is located in the `Visual Studio` subfolder of the `Samples` path.

#### 5.15.5 VStudio.bld Sample

This sample demonstrates an entire build process that uses Microsoft Visual Studio 2008 thru 2015.

The build recursively registers the components, builds several Visual Studio projects, and increments a build number. Custom actions are used to build the projects, which will prevent unnecessary rebuilds, increment versions when building, set project base addresses, etc.

If any step fails, the failure steps are executed, which send an e-mail with the error output from the failed step. You may need to update the e-mail address and modify the server settings. The `SMTP_SERVER`, `TO_EMAIL`, and `FROM_EMAIL` project macros should be updated to appropriate values, and a username and password assigned to the Send Mail steps if necessary.

The first time the project is built, all the projects are built; if built again (close and reopen the project or reset the build status and build), VisBuildPro detects that they are up-to-date and none of the projects get built.

Several standard system macros are used, such as `PROJDIR`, to generically determine the file locations, `FAILSTEP_OUTPUT` to show the error message from a previous step, `DOSCMD` to execute operating system commands, `DATETIME` to log the current date/time. A `LOGFILE` project macro is also defined to log all build output to `VStudio.log` in a temporary directory.

*Note:* This sample project is located in the `Visual Studio` subfolder of the `Samples` path.

### 5.16 Miscellaneous

#### 5.16.1 SaveStatus.bld Sample

Demonstrates saving the build status with the project for continuation from the failure point in a build after the project has been closed and reopened.

*Note:* This sample project is located in the `Misc` subfolder of the `Samples` path.

### 5.16.2 TestVFP.bld Sample

Demonstrates building a Visual FoxPro project. Use the VFP wizard to create a new project called TestVFP in the same directory as TestVFP.bld, then build. To view the BuildVFP script subroutine that is used, double-click on the project step, then click on Script Editor in the step properties dialog and go to the Project tab.

*Note:* This sample project is located in the `Misc` subfolder of the `Samples` path.

### 5.16.3 WebLauncher Sample

This is a sample ASP.NET 2.0 (and 1.1) web project that can be used for performing builds on a remote computer. The web page prompts for the project file to build, whether to wait for completion (build synchronously and log build output to web page or return immediately after starting build), and any macros to pass to the project, and then starts the specified build on the computer hosting the ASP.NET web app. This sample can be tweaked to provide custom inputs for specific builds, etc.

*Notes:*

- The sample projects are located in the `Misc\WebLauncher` subfolder of the `Samples` path.
- The code in the `WebLauncher` folder is the ASP.NET v2.0 version, and the `NET1.1` subfolder contains the ASP.NET v1.1 version.
- The `Setup.bld` project can be built to create a virtual directory for the project and build the project -- it must be run as an (elevated) administrator user on Windows Vista and later.
- The *identity* element in the web site's `web.config` file must be configured with an appropriate impersonation account *username* and *password* having the necessary rights to perform a build.

## 5.17 Advanced

### 5.17.1 VisBuildPro Samples

The projects in the `Samples\VisBuildPro` folder are used to build Visual Build. They use a custom GUI front-end, several custom actions, and also demonstrate some advanced build techniques.

The `VisBuildPro.bld` sample uses subroutines and conditional build rules to dynamically include/exclude parts of the build. Defaults are defined by several project macros. These macros are used in build rules in the project to determine which steps get built or skipped. The project is modularized by subroutines, some that call other subroutines.

The BuildLauncher C# project (`BuildLauncher\BuildLauncher.csproj`) prompts for the inputs needed for the build (defining appropriate defaults) and launches `VisBuildPro.bld` with the values that the user selects.

Other capabilities, such as updating VC6 build directories in the registry, conditionally building a group of steps by comparing the size/timestamp of two files, and using the Process Files, FTP, and Telnet actions, are also demonstrated.

The `Backup.bld` sample demonstrates backing up application configuration data, registry keys, and performing an incremental backup of files that have changed since last copied.

### 5.17.2 ObjectModel Samples

The ObjectModel samples demonstrate providing a custom UI for inputting values for a build. They accept values required for a build, and kick off the Visual Build project (using the Visual Build object model or command-line interface) after the values have been entered.

The samples are located in the `Samples\ObjectModel` folder. Visual Basic 6 (VBClient), VB.NET (VBNetClient), and C# (CSharpClient) versions are provided. The `VisBuildPro` sample also provides



the BuildLauncher project, demonstrating the front-end used for starting builds of Visual Build.

### 5.17.3 VBPLogger Sample

The Visual Build logging architecture supports registration of custom logging components. Two sample loggers (C# and VB6) show how to create a custom logging component to replace the built-in logging components. They write to the log file in a custom XML format.

The samples are located in the `Samples\Misc\VBPLogger` folder. To use:

1. Build the logger project by opening either `VBPLogger.csproj` (C# version) or `VBPLogger.vbp` (VB6 version) and building.
2. Open `Register.bld` and build the related Register step to register the associated logger component with VBP.
3. Go to *Tools | Application Options | Logging* and change the format to VBPLogger.
4. Go to *File | Properties | Logging* and delete the log file.
5. Rebuild the `Register.bld` project to generate a log using the custom logger.

To debug, open the project in Visual Studio, set `VisBuildPro.exe` as the debug executable, and debug the project in the IDE.

Visual Build loggers are COM components implementing a standard interface (`ILogger`, defined in the `VisBuildSvr.dll` type library) and custom-registered for Visual Build to locate them. This is done by adding the logging component's ProgId in a registry key under `HKEY_LOCAL_MACHINE\SOFTWARE\Kinook Software\Visual Build Professional 9\Logging\<Format>`. During a build, all registered loggers for the log format selected in the Logging options dialog are instantiated and initialized. To implement logging, a logging component connects to the build events fired by the Builder object and logs any pertinent information when the events occur.

The logging architecture supports multiple logging components for each file format, so a custom format and logger could be defined, or the standard logging component for a given format could be used for logging, and a custom logging component could be used for extended logging functionality (for instance, to generate a summary log file or to write additional information to the main log file). As an example of writing to a different file, the custom logging component could append 'Summary' to the value of the LOGFILE macro, or it could use a different macro that its log filename could be placed in, and only write failed steps or generate HTML log output. See the VBPLogger sample for more details.

*Note:* Some of the interfaces defined in the `VisBuildSvr` type library are marked as hidden; to view the hidden interfaces in the VB Object Browser, right-click and choose Show hidden members.

## 6 Commands and Procedures

### 6.1 Automated Builds

Visual Build provides several options for automating builds.

1. Integration with the Windows Scheduled Tasks / Task Scheduler service is provided for quickly setting up scheduled builds.
2. Support for continuous builds is available by using a repeating build rule.
3. The GUI App provides a command-line interface. The graphical Windows application is launched to open and build the project.
4. A Console App also provides a command-line interface (with nearly identical syntax), which can be used to perform a build without displaying a user interface or requiring interactive desktop access.
5. A project can be launched on a remote server by using the Remote tabs of the `VisBuildPro` Project or Run Program actions. Also see the `Server.bld` sample for more details.

6. Remote builds can also be initiated via a web interface. See the WebLauncher sample.
7. The application and builder objects are exposed as COM automation servers, so you can write script or code in any language that supports COM to automate a build.

*Note:* You can use the VisBuildPro Project action to launch the GUI App or Console App from within a build.

Each method actually uses the COM server components behind the scenes, so the build behavior will be the same regardless of the method used. All components honor the settings configured in Application options. The only difference is that the GUI App may prompt for input (for instance, if an undefined macro is encountered or upon exit if the project has been modified), the Console App never prompts for input and any changes made to the project will be discarded unless explicitly saved in the build, and code that utilizes the COM objects will need to provide their own prompting if required.

The ability to save the build status with the project can be useful during automated builds. By adding a Run Script steps which saves the build status with the project on failure, if the automated build is launched again on that project, it will continue from the point of failure. It is also advised to add a step which saves the project **without** the build status on a successful build so that the next time the build is launched, the entire build will be performed (otherwise, all steps would be marked completed and no steps would be built). Also, ensure that the project file is writeable before saving, or the save step will fail. See the SaveStatus.bld sample for an example of using this capability.

### 6.1.1 Console App

The Visual Build console app can be used to perform automated builds. Unlike the GUI App, the console app:

- Does not require interactive desktop access or a currently logged on user (but third party tools called in step actions may themselves require interactive desktop access in order to run properly).
- Does not prompt for input if an undefined macro is encountered.
- Discards any changes made to the project via the object model unless explicitly saved in the build.

```

Windows PowerShell
PS D:\> & 'C:\Program Files\VisBuildPro8\VisBuildCmd.exe' D:\logging.bld
VisBuildCmd (x64), Version 8.7.0.2
Copyright (C) 1999-2014 Kinook Software, Inc. All rights reserved.
Registered to: Kinook Software, Inc. (1-computer license)

11/19/2014 7:09:44 AM: Starting Build: 'D:\logging.bld'
11/19/2014 7:09:44 AM: Building project step 'Project steps'...
11/19/2014 7:09:44 AM: Building project step 'normal'...
normal
11/19/2014 7:09:44 AM: Building project step 'error'...
error
11/19/2014 7:09:44 AM: Building project step 'warning'...
warning
11/19/2014 7:09:44 AM: Building project step 'detailed'...
detailed
11/19/2014 7:09:44 AM: Building project step 'diagnostic'...
11/19/2014 7:09:44 AM: Build successfully completed (elapsed = 00:00:00).
PS D:\>

```

The console app supports the following syntax.

```

VisBuildCmd.exe ["MACRO=VALUE"] [/nologo] [/nooutput] [/loglevel level]
[/logfile "filename"] [/profile "profile"] [/pwd "password"] [/configpath
"path"] [/config "config file"] [/macros "macro file"] [/script "script
file"] [/steps "step file"] [/mta] [/b] "[Drive+Path\]ProjectFile.bld"

```

*Notes:*

- Parameters in square brackets are optional.
- See the Automatic Setup topic for scheduling builds with the necessary command-line flags.
- To hide the console build window, `HideConsole.exe` (installed in the same folder as `VisBuildCmd.exe`) can be used by prefixing `HideConsole.exe` to the `VisBuildCmd.exe` command-line call.

"MACRO=VALUE" – defines one or more temporary macro values. Special characters within macros values should be escaped.

`/nologo` – Suppresses display of application version and copyright.

`/nooutput` – indicates silent mode for the build (does not echo build output to Std Output); if logging is enabled in Application Options, all build output will be logged as specified.

`/loglevel` – determines which level of log messages will be logged to Std Output when building a project (does not apply with the `/nooutput` option). Overrides the default log level defined in application options. Messages with a level the same or lower than the level specified here will be logged.

`/logfile` – overrides the log file to use for the build. Can be a filename or "" to disable file logging.

`/profile` – sets the build profile to use for the build (defines a `_BUILD_PROFILE_` temporary macro with the specified value). *Note:* The profile can also be updated in the `vbuild_ProjectLoaded` script event, which overrides this value.

`/pwd` – specifies a user-defined key for decrypting protected properties in the project file.

`/configpath` – Specifies an alternate path to load all configuration files and user actions from (takes precedence over the flags below). If not specified, the default configuration files path will be used.

`/config` – Specifies an alternate configuration file (which holds Application Options) to use. If not specified, the `VisBuildPro.config` file in the configuration files path will be used.

`/macros` – Specifies an alternate global macros file to use. If not specified, the `VisBuildPro.macros` file in the configuration files path will be used.

`/script` – Specifies an alternate global scripts file to use. If not specified, the `VisBuildPro.Global.scripts` file in the configuration files path will be used.

`/steps` – Specifies an alternate global steps file to use. If not specified, the `VisBuildPro.steps` file in the configuration files path will be used.

`/mta` – Specifies that the Builder component and the build thread get created in the MTA.

`/b` – opens and builds the specified project (optional, for compatibility with GUI App).

"ProjectFile.bld" – indicates the Visual Build project to open and build. If the build succeeds, Visual Build exits with a (0) success code; if the build fails, one of the following codes will be returned to the operating system. *Note:* This flag must immediately precede the filename, and all other flags or macros must come before it.

*Note:* Command-line arguments can also be provided in a response file using the syntax `VisBuildCmd.exe "@[Drive+Path\]ResponseFile.ext" . . .`. Each response file specified will be processed for its command-line arguments. The response file must contain one parameter per line with no surrounding double quotes.

Exit code	Meaning
0	Success
1	Error occurred building the last step
2	Build aborted by user
3	Error opening project file or file not found
4	Error parsing command-line
5	Exception occurred
6	Unexpected error

## 6.1.2 GUI App

The Visual Build GUI app can be started from the command-line. It supports the following syntax (or use the VisBuildPro Project action to help configure the command-line to use):

```
VisBuildPro.exe ["MACRO=VALUE"] [/s] [/d] [/f] [/logfile "filename"]
[/profile "profile"] [/pwd "password"] [/configpath "path"] [/config
"config file"] [/macros "macro file"] [/script "script file"] [/steps "step
file"] [/mta] [/b] ["[Drive+Path\]ProjectFile.bld"]
```

### Notes:

- Parameters in square brackets are optional, and the project filename must be the last parameter.
- See the Automatic Setup topic for scheduling builds with the necessary command-line flags.
- Use the Create Shortcut command to configure a shortcut with command-line flags to open the current project.

"MACRO=VALUE" – defines one or more temporary macro values. Special characters within macros values should be escaped.

/s – indicates silent mode for the build (does not display message boxes or dialog boxes, prompt for input when encountering an undefined macro, or stop at breakpoints, and discards any changes to the project).

*Note:* Use this flag to cause the chained instance to exit on build failure.

/d – pauses the build at the first step of the project for debugging (cannot be used with silent mode).

/f – don't close Visual Build if the command-line build fails (applies only with silent mode).

/logfile – overrides the log file to use for the build. Can be a filename or "" to disable file logging.

/profile – sets the build profile to use for the build (defines a `_BUILD_PROFILE_` temporary macro with the specified value). *Note:* The profile can also be updated in the `vbuild_ProjectLoaded` script event, which overrides this value.

/pwd – specifies a user-defined key for decrypting protected properties in the project file.

/configpath – Specifies an alternate path to load all configuration files and user actions from (takes precedence over the flags below). If not specified, the default configuration files path will be used.

/config – Specifies an alternate configuration file (which holds Application Options) to use. If not specified, the `VisBuildPro.config` file in the configuration files path will be used.

/macros – Specifies an alternate global macros file to use. If not specified, the `VisBuildPro.macros` file in the configuration files path will be used.

/script – Specifies an alternate global scripts file to use. If not specified, the

VisBuildPro.Global.scripts file in the configuration files path will be used.

/steps – Specifies an alternate global steps file to use. If not specified, the VisBuildPro.steps file in the configuration files path will be used.

/mta – Specifies that the Builder component and the build thread get created in the MTA.

/b – tells Visual Build to open and build the specified project. If the build succeeds, Visual Build exits with a (0) success code; if the build fails, one of the following codes will be returned to the operating system. *Note:* This flag must immediately precede the filename, and all other flags or macros must come before it.

"ProjectFile.bld" – indicates the Visual Build project to open

*Note:* Command-line arguments can also be provided in a response file using the syntax VisBuildPro.exe "@[Drive+Path\]ResponseFile.ext" ... Each response file specified will be processed for its command-line arguments. The response file must contain one parameter per line with no surrounding double quotes.

Exit code	Meaning
0	Success
1	Error occurred building the last step
2	Build aborted by user
3	Error opening project file or file not found
4	Error parsing command-line
5	Exception occurred
6	Unexpected error

*Note:* The *honor command-line switches* user option determines whether these flags 'stick' if a build is canceled/failed and then continued.

To quickly launch Visual Build and create a step for a filename that has an associated action, this syntax can be used:

```
VisBuildPro.exe "[Drive+Path\]Filename.ext"
["[Drive+Path\]ProjectFile.bld"]
```

### 6.1.3 Scheduling Builds

The Windows scheduling services can be used to perform scheduling builds to run once at a specific time or on a regular basis. This topic describes the steps to configure a scheduled build. It is a good idea to enable file logging so that, if necessary, the build output can be examined for any errors after the build has completed. The console or GUI command-line apps can be used for scheduled builds; the GUI app requires interactive desktop access, while the console app does not.

*Note:* When initially configuring a scheduled build, the build may function correctly when launched interactively but fail to build from the scheduling services. This is usually because the user account that the scheduled build runs under does not have the necessary rights to open the project file, access computer or network resources, etc. If this occurs, the application may also be unable to display any error message or dialog to the user. In this situation, check the Windows Application **Event Viewer** (available in Administrative Tools), as any errors during startup of the console or GUI app will be written to the Event Log (if the necessary rights are available).

#### Automatic Setup

## Manual Setup

### 6.1.3.1 Automatic Setup

Two methods are available for quickly creating a Scheduled Task for scheduling builds:

- 1) Choose *Create Scheduled Task* on the Tools menu. A new scheduled task is created for the current project, with all existing temporary macros added as command-line parameters for the call.
- 2) Click the *Create Scheduled Task* button from the VisBuildPro Project action. A new scheduled task is created and the command-line flags set according to the values configured in the step.

After the task is created, the task's property pages are displayed.

#### **Important:**

- For *Scheduled Tasks* (Windows 2000, XP, 2003), in order for the task to successfully run, the user's password must be entered by clicking the Set Password button on the Task tab. If this is not done, the task will not be able to start.
- For *Task Scheduler* (Windows Vista and later), you may wish to adjust the credentials on the task's General tab (by default, the task is configured to run as the interactive user only when the user is logged on) and configure other conditions and settings on the associated tabs.

By default, the task is scheduled to run daily, starting three minutes after the task was created; this schedule can be modified on the Schedule tab for Scheduled Tasks and the Triggers tab for Task Scheduler (builds can also be scheduled to run monthly, weekly, once, at startup, when idle, on multiple schedules, etc.). After entering and confirming the account password and making any scheduling changes, click OK to accept the changes. The Scheduled Tasks or Task Scheduler window will then be displayed, showing the new task and any other existing tasks. To run the task immediately, right-click on the task on choose Run, or wait for the scheduled time to pass and the task will be started and the project loaded and built.

#### *Notes:*

- When created from the Tools menu, a user option determines whether the GUI or Console app is used. When created from the VisBuildPro Project action, the action settings determine which executable is launched. To launch the Console App instead of the GUI App, change `VisBuildPro.exe` in the Run field to `VisBuildCmd.exe`.
- The Windows Task Scheduling service must be started in order for Scheduled Tasks to run.
- The file `SchedLgU.Txt` in the Windows, WINNT, or Windows\Tasks folder can be used to view the launch and completion status of each scheduled task.
- The account the scheduled task runs under may not have the necessary permissions to network shares and other network resources network that might be accessed during a build. One option for resolving share permissions is to add a step which maps a network drive using the credentials of a user with appropriate rights, then reference that drive in build steps, and disconnect the network drive at the end of the build.
- On Windows Vista and later, programs (even those started by an administrator) do not run with administrative privileges and may not have the necessary rights for building (registering, copying files, modifying the registry, etc.). To avoid this problem, configure the task to run under an administrator account, and on the General tab of the Task properties dialog, check the *Run with highest privileges* checkbox. Another option is to disable UAC.

### 6.1.3.2 Manual Setup

To create a task manually, edit an existing task, or use the AT service to schedule a build, follow the instructions below.

#### **Task Scheduler (Windows Vista and later)**

1. To start the Task Scheduler, click *View Scheduled Tasks* from the Visual Build Tools menu (or select *Start | All Programs | Administrative Tools | Task Scheduler* in Windows).
2. Click *Create Basic Task...*
3. Type a name for the task and choose when it will be run.
4. Select an action of *Start a program*.
5. Browse to the Visual Build installation directory (typically `C:\Program Files\VisBuildPro9`) and select `VisBuildPro.exe` (GUI App) or `VisBuildCmd.exe` (console app).
6. In the *Add arguments* field, enter `/s /b "<full path>\<projectfile>.bld"`.
7. Optionally, check *Open the Properties dialog for this task when I click Finish*.
8. Configure other settings on the task properties dialog as needed.
9. Visual Build will be started at the scheduled time, build the specified project, and close. The `/s` flag causes Visual Build to close without prompting even if an error occurs.

### Scheduled Tasks (Windows 2000, XP, 2003)

1. To start Scheduled Tasks, click *View Scheduled Tasks* from the Visual Build Tools menu (or select *Start | Settings | Control Panel* from the taskbar and double-click on *Scheduled Tasks*).
2. Start the Scheduled Task Wizard by double-clicking on *Add Scheduled Task*.
3. Select Visual Build Professional from the list when prompted for the program you wish to run.
4. Type a name for the task and choose when it will be run.
5. Enter the username and password that the application should be executed under.
6. Check *Open advanced properties for this task when I click Finish*.
7. To use the console app, change `VisBuildPro.exe` to `VisBuildCmd.exe` in the *Run* field.
8. Edit the *Run* field and add `/s /b "<full path>\<projectfile>.bld"` to the field for the GUI App or `"<full path>\<projectfile>.bld"` for the Console App.
9. Visual Build will be started at the scheduled time, build the specified project, and close. The `/s` flag causes Visual Build to close without prompting even if an error occurs.

### AT Service (Windows 2000 and later)

1. Open a Command Prompt.
2. Enter an AT command to schedule Visual Build. If the Console App is used, the build will run totally silent (will not display a GUI). For example, to build the `RegEdit.bld` project at 2:32pm today (the `/INTERACTIVE` flag is not necessary when using the GUI app):
 

```
AT 14:32 c:\Progra~1\VisBuildPro9\VisBuildCmd.exe \"c:\My
Projects\xyz.bld\"
AT 14:32 /INTERACTIVE c:\Progra~1\VisBuildPro9\VisBuildPro.exe /s /b
\"c:\My Projects\xyz.bld\"
```
3. Visual Build will be started at the scheduled time, build the specified project, and close. The `/s` flag causes Visual Build to close without prompting even if an error occurs.

#### Notes:

- If Visual Build returns an exit code of `0x3` when called from a scheduled build, the user that the build is running under may not have rights to open and load the project (`.bld`) file. Ensure that the user has the necessary rights.
- When using the AT service, the path to `VisBuildPro.exe` or `VisBuildCmd.exe` must be the short filename (no spaces). Add the `/next:<day of month/day of week>` flag to schedule a build for a future date, or add the `/every:<day of month/day of week>` flag to schedule a recurring build. For example, to build the project on the 15th day of the month:
 

```
AT 14:32 /NEXT:15 c:\Progra~1\VisBuildPro9\VisBuildCmd.exe \"c:\My
Projects\xyz.bld\"
```
- Some steps (for instance, if a Run Script action calls script which displays a messagebox) or other programs called from Visual Build may themselves require interactive desktop access to build properly. In these cases, the `/INTERACTIVE` switch will be needed even with the Console app.

## 6.1.4 Continuous Integration

### Within Visual Build

The ContinuousIntegration.bld sample shows how to implement continuous integration builds (automatic builds whenever source code changes are submitted to source control) with Visual Build.

#### BuildBot

Continuous integration can be implemented via BuildBot by invoking the Visual Build command-line interface. In BuildBot, add a ShellCommand like this:

```
from buildbot.plugins import steps

f.addStep(steps.ShellCommand(command=["C:\Program
Files\VisBuildPro9\VisBuildCmd.exe", "/b", "ProjectFile.bld"]))
```

#### Jenkins/Hudson

Continuous integration can be performed via Jenkins/Hudson by invoking the Visual Build command-line interface.

1. In Jenkins/Hudson, create a new job to *Build a free-style software project*.
2. Add a build step of *Execute Windows batch command* with a command like:  
"C:\Program Files\VisBuildPro9\VisBuildCmd.exe" /b "C:\Program Files\VisBuildPro9\Samples\RegEdit.bld"

#### TeamCity

Continuous integration can be implemented via TeamCity by invoking the Visual Build command-line interface.

1. In TeamCity, create a new project.
2. Create a new build configuration with a Runner Type of *Command Line*, a Working directory of the path to the .bld file, a Command executable of C:\Program Files\VisBuildPro9\VisBuildCmd.exe, and Command parameters of /b ProjectFile.bld.

#### Team Foundation Build

Visual Build projects can also be called from Team Foundation Build by invoking the Visual Build command-line interface from an MSBuild Exec task (2008 and earlier). Add code like this to the end of the TFSSBuild.proj file (see this article for more details on modifying and running the Team Build build definition):

```
</ItemGroup>
  <Target Name="AfterCompile">
    <Exec Command="'C:\Program Files\VisBuildPro9\VisBuildCmd" /b
"$(SolutionRoot)\MyProject.bld" ' />
  </Target>
```



```
</Project>
```

In addition to the *AfterCompile* target, several additional targets are available for specifying when Visual Build should be called. And the *BeforeBuild* and *AfterBuild* targets can also be used from regular Visual Studio projects (outside of Team Foundation Build).

For Visual Studio 2010 and later, use the *InvokeProcess* activity (FileName -> %ProgramFiles%\VisBuildPro9\VisBuildCmd, Arguments -> /b "\$(SolutionRoot)\MyProject.bld").

## CruiseControl .NET

Continuous integration can be implemented via CruiseControl .NET by invoking the Visual Build command-line interface from an Executable task in the CCNet.config file. For example:

```
<cruisecontrol>
  <project>
    <name>Sample</name>
    <triggers />
    <tasks>
      <exec>
        <executable>C:\Program Files\VisBuildPro9\VisBuildCmd.exe</executable>
        <baseDirectory>C:\Program Files\VisBuildPro9\Samples</baseDirectory>
        <buildArgs>/b RegEdit.bld</buildArgs>
        <buildTimeoutSeconds>1000</buildTimeoutSeconds>
      </exec>
    </tasks>
  </project>
</cruisecontrol>
```

## 6.2 64-bit vs. 32-bit

Visual Build (the GUI and Console apps and COM object model) is available in 32-bit and 64-bit editions. The **32-bit** edition is compatible with 32-bit editions of Windows, and also 64-bit editions of Windows via the WoW64 subsystem. The **64-bit** edition is a native x64 executable that does not require the WoW64 subsystem and can only be installed on a 64-bit edition of Windows.

### File System

The 32-bit edition of Visual Build is able to call 64-bit executables (and vice versa), but Windows does implement file system redirection from %WINDIR%\System32 to %WINDIR%\SysWOW64 for 32-bit programs. Use the 64-bit edition of Visual Build, or when using the 32-bit edition use %WINDIR%\Sysnative instead of %WINDIR%\System32 to run 64-bit programs in this folder or to pass this folder to another 32-bit program and prevent redirection (on Windows 2003, first install the Windows hotfix that adds support for Sysnative). Other options are also available for calling programs in the true 64-bit %WINDIR%\System32 folder from the 32-bit edition of Visual Build:

1. Manually copy executables from %WINDIR%\System32 to another folder (using the 64-bit Explorer.exe or cmd.exe) to call them from Visual Build.
2. Run SUBST S: %WINDIR%\System32 from a 64-bit Command Prompt and then access the folder via S: from Visual Build.
3. Create an NTFS junction point that points to the %WINDIR%\System32 folder. Access the NTFS junction point instead of %WINDIR%\System32.

#### Notes:

- To call the 64-bit version of cmd.exe from the 32-bit edition of Visual Build when using the DOSCMD system macro or the *Command to run before main command field* on the advanced tab of the Run Program and derived actions, create a COMSPEC global macro with a value of

`%WINDIR%\Sysnative\cmd.exe`.

- To call the 32-bit version of `cmd.exe` from the 64-bit edition of Visual Build when using the `DOSCMD` system macro or the *Command to run before main command field* on the advanced tab of the Run Program and derived actions, create a COMSPEC global macro with a value of `%WINDIR%\SysWOW64\cmd.exe`.

## Registry

On 64-bit editions of Windows, some portions of the registry are redirected or reflected for 32-bit applications. Options for accessing the 64-bit registry view from the 32-bit edition of Visual Build:

1. Use the *Access 64-bit registry view* option of the Read Registry and Write Registry actions.
2. From script code, call the `vblid_ReadRegHKLM64` system script function.
3. Use registry locations that are shared by 32- and 64-bit applications.
4. Call the 64-bit `%WINDIR%\Sysnative\reg.exe` from a Run Program action (see File System section above).

To access the 32-bit registry view from the 64-bit edition of Visual Build:

1. Use the *Access 32-bit registry view* option of the Read Registry and Write Registry actions.
2. Use registry locations that are shared by 32- and 64-bit applications.
3. Call the 32-bit `%WINDIR%\SysWOW64\reg.exe` from a Run Program action (see File System section above).

## COM Object Model

The Visual Build object model is based on COM and is implemented as both 32-bit and 64-bit components. To call the object model from 64-bit applications or ASP.NET on 64-bit Windows, the 64-bit edition of Visual Build must be installed or the 32-bit version of ASP.NET must be switched to.

## 6.3 Configuration Files

Visual Build (both the GUI and Console apps) utilizes the following configuration files for global application settings and user action files:

- `VisBuildPro.config`: Application options
- `VisBuildPro.macros`: Global macros
- `VisBuildPro.Global.scripts`: Global scripts
- `VisBuildPro.steps`: Global subroutine steps

The File Locations dialog displays the current location of the config files being used. The default path used for the configuration files is determined at startup as follows:

- First, if the `/configpath` GUI or console command-line switch is specified, that path will be used for loading and saving the configuration files.
- Next, if the `REG_SZ` registry value `HKEY_LOCAL_MACHINE\SOFTWARE\Kinook Software\Visual Build Professional 9\ConfigFilesPath` exists, the path in that value will be used for loading and saving the configuration files.
- If neither of the above exist, the configuration files will be loaded and saved in the Windows common Application Data folder (located at `%ALLUSERSPROFILE%\Application Data\Kinook Software\Visual Build Professional 9` -- usually `C:\Documents and Settings\All Users\Application Data\Kinook Software\Visual Build Professional 9` on Windows 2003 or `C:\ProgramData\Kinook Software\Visual Build Professional 9` on Windows Vista and later).

Two additional methods are provided for explicitly specifying where configuration files are loaded from and/or saved to:

- The location of each individual configuration file can be overridden by supplying additional command-line flags when launching the Visual Build GUI or Console apps.
- Configuration files can be explicitly loaded from a Run Script step in a build using the Object Model:

```
Application options => Application. Options. Load          "drive:path\to\filename.ext"
Global macros => Application. Macros(vbldMacroGlobal) .Load
"drive:path\to\filename.ext",          True
Global scripts => Application. Scripts(vbldScriptGlobal) .Load
"drive:path\to\filename.ext"
Global subroutine steps => Application. Project. Steps(vbldStepGlobalSubroutine) .
Load          "drive:path\to\filename.ext"
```

## 6.4 Project File Format

Visual Build projects are stored in files with a `.bld` or `.bldx` extension.

When saved with a `.bld` extension, the project file is stored in XML format with UTF-8 encoding (including a BOM [byte order mark] if the related option is enabled), which can be stored as text in a source control system.

When saved with a `.bldx` extension, the project file is stored in a compressed format and encrypted with 256-bit AES encryption. When specifying an edit password, it is more secure to save the file compressed and encrypted with a `.bldx` extension to prevent the edit password from being removed from the `.bld` file. When specifying a project password, the project password is also used as the key for encrypting the `.bldx` file (otherwise a hard-coded encryption key is used), providing additional security for the content of the project file.

## 6.5 GUI Features

### 6.5.1 Copy and Paste

Visual Build supports clipboard operations in the following ways:

- To copy the selection in the step properties or macro properties dialog with macros expanded, type Ctrl+M. To copy with macros and script expanded, type Ctrl+Shift+M.
- Select one or more steps or macros and copy (Ctrl+C) or cut (Ctrl+X) them to the clipboard. The selected items are copied to the clipboard in text format as XML. If a selected step is collapsed, its child steps are also copied or cut.
- Steps or macros on the clipboard can be pasted (Ctrl+V) into other step or macro positions within the same or other instances of Visual Build. Steps are inserted after the selected step when pasting; if a macro of the same name and type already exists, the copied macro's name is suffixed with " - Copy x" to make it unique.
- Copy an action from the Actions pane and paste it into a step pane to create a new step for the action.
- Items placed on the clipboard can also be pasted into other applications such as text editors or email applications, and then pasted back into instances of Visual Build.
- The contents of (the entire file or individual steps or macros) Visual Build project files can also be copied from a text editor into instances of Visual Build.

## 6.5.2 Create Desktop Shortcut

To create a desktop shortcut for the current project file, choose *Create Desktop Shortcut* on the Tools menu. A shortcut is created on the desktop to open the current project, with all existing temporary macros added as command-line parameters for the call. The shortcut can be double-clicked to open the project.

Notes:

- To add additional command-line flags, right-click on the shortcut, choose properties, edit the Target field, and click OK.
- To launch and build the project with the Console App instead of the GUI App, change `VisBuildPro.exe` in the shortcut's Target field to `VisBuildCmd.exe`

## 6.5.3 Drag and Drop

Visual Build supports drag and drop in the following ways:

- Drag and drop an action from the Actions pane onto a step pane to create a new step for the action.
- Select one or more steps in a project and drag and drop them to move or copy within a project. By default, the steps are moved; to copy the selected steps, hold down the Ctrl key before dropping. The steps are inserted after the step they are dropped onto. If a selected step is collapsed, its child steps are also moved or copied. Right-drag to display a context menu with Copy, Move, and Cancel choices on drop.
- Select one or more steps in a project and drag and drop them to another instance of Visual Build. By default, the steps are copied; to move the selected steps, hold down the Shift key before dropping. Right-drag to display a context menu with Copy, Move, and Cancel choices on drop.
- Drag and drop the column headers in the Step Panes to rearrange the column order.
- Drag and drop a .bld file from Windows Explorer onto Visual Build to open the project.
- Drag and drop a file (with an extension configured for a custom action) from Windows Explorer onto Visual Build to create a new step for that action.

## 6.5.4 Explorer Interface

Visual Build integrates with the Windows Shell in the following ways:

- Double-click a file with a .bld extension from Windows Explorer to open it in Visual Build.
- Right-click a file with a .bld extension from Explorer, and choose Open to open it in Visual Build, or choose Build to open and build the project.
- Drag a Visual Build project file from the Windows Explorer and drop it on a Visual Build window to open the project.
- Drag and drop a project file onto a Visual Build shortcut to open the file in a new Visual Build instance.
- To create a new Visual Build project from Explorer, click File|New|Visual Build Professional Project from the menu, or right-click and choose New|Visual Build Professional Project.
- To start Visual Build from a Command/DOS Prompt, type `start VisBuildPro` and press Enter; to open a project, type `start [Drive+Path\]ProjectFilename.bld` and press Enter.
- Drag and drop a filename onto Visual Build to create a new step with the specified action for that file extension.
- To start Visual Build and insert a custom action for a given filename, open a Command/DOS Prompt, type `start [Drive+Path\]Filename.ext` and press Enter.

## 6.5.5 Multiple Selection

Visual Build supports multiple selection of macros and steps for copying, moving, deleting, indenting, and editing properties of multiple steps at once.

To multi-select items

- With the mouse:
  - Hold the *Ctrl* key while clicking individual items to select or unselect that item
  - Click to select the first item, then hold the *Shift* key and click the last (to select a contiguous range)
- With the keyboard:
  - Hold the *Ctrl* key down while using the arrow keys to navigate to different items, and press the Space bar to select or unselect and individual item
  - Use the arrow keys to navigate, holding the *Shift* key down while navigating to select a contiguous range

## 6.5.6 Navigation

### Forward/Back

The Visual Build GUI app remembers each step and macro that the user has selected and allows navigating forward and backward to these items, similar to navigation in a web browser. Forward/back functionality is accessed via the Go menu, the view toolbar, the associated keyboard shortcuts, or the Forward/Back dedicated keys or Forward/Back mouse buttons on a 5-button mouse.

*Note:* By default, the last 500 positions are tracked for forward/back. To adjust this limit, start RegEdit and create/edit a DWORD value named `MaxForwardBackHistory` under `HKEY_CURRENT_USER\Software\Kinook Software\Visual Build Professional 9\Options` (set the value to 0 for unlimited history).

### Step Navigation

- To quickly show the step that is currently being built or debugged, choose *Go | Current Step* on the menu bar.
- To jump to the subroutine step for a Subroutine Call step, choose *Go | Subroutine* on the menu bar.
- To quickly navigate to a particular step in the current step pane, use *Go | To Step* on the menu.

### Build Output Navigation

- When an error occurs during a build, Visual Build can open the Script Editor or Step Properties dialog to pinpoint the cause of error. This is accomplished by choosing *Go | Last Error* on the menu bar.

*Note:* This feature may not work reliably for script engines that do not fully support Windows Active Scripting. VBScript and Jscript work properly, but the PerlScript, Python, and RubyScript engines do not provide full error information.

- You can quickly navigate between the output for a step in the Output pane and the related step in a Step pane (and vice versa) via the *Go To | Step Output* menu item.
- Visual Build provides an easy way to navigate from a compiler error or filename listed in the build output to the associated source code file by selecting the filename or line containing the compiler error, and choosing an item from the *View | Shell* menu to open the file in the configured viewer, its

associated application, or a folder in Windows Explorer. When a filename and line number is displayed in the Output pane in the format `C:\Path\To\Filename.ext(999) : optional error description`, and if the configured viewer application supports a command-line flag for opening a file at a specific line number, the file will be opened in the viewer at the specified line number. See the General options dialog topic for details on configuring a supported viewer.

## External Launching

A filename or folder in any step or macro property can also be quickly *opened* in a viewer, *launched* in its associated application, opened in *Explorer* (if a filename is selected, its containing folder will be opened), or opened in a Command Prompt by clicking the Shell button on the Step or Macro properties dialogs or by right-clicking in the Output or Step panes and choosing one of the drop-down choices (in a Step pane, click Shell on the menu for actions whose default property is a filename or folder).

### 6.5.7 Tool Tip Features

Visual Build uses tool tips extensively to aid in creating and debugging steps and macros. Tool tips are displayed when the mouse cursor is held over a field for a couple seconds.

**Step Panes, Macros Pane, Properties Pane, and Watches Pane:** If a column contains macros or is longer than the column width, the cell value with all macros expanded is shown in a tool tip (if the field contains a script expression and the Shift key is held down when hovering, the script code is also evaluated and displayed in the tool tip). For the Build Rule column, the build rule is displayed (hovering while holding Shift down will display the result of the rule evaluation [true or false]). When scrolling the view, the name of the top item is shown in a tool tip.

**Step Properties and Macro Properties dialogs:** If a field contain macros, the value with all macros expanded is shown when the mouse is held over each field (if an undefined macro is found in the field, it is left unexpanded). If a field contains a script expression, if the Shift key is held down, the script code is also evaluated and displayed in the tool tip. To copy the selection to the clipboard with macros expanded, type Ctrl+M; to copy with macros and script expanded, type Ctrl+Shift+M.

**Insert Macro Dialog:** The macro's description is displayed in a tool tip when the mouse is held over a macro in the list (the description, value, and type can also be displayed in columns in the dialog).

*Note:* Tool tips are not displayed for encrypted properties.

### 6.5.8 Undo/Redo

The Visual Build GUI app provides unlimited undo and redo capabilities when modifying steps and macros. Undo/redo functionality is accessed via the Edit menu or the standard toolbar.

*Notes:*

- By default, the last 10,000 changes are tracked for undo/redo. To adjust this limit, start RegEdit and create/edit a DWORD value named `MaxUndoHistory` under `HKEY_CURRENT_USER\Software\Kinook Software\Visual Build Professional 9\Options` (set the value to 0 for unlimited undo/redo history).
- The script editor also provides undo/redo while editing script code (this is maintained in a separate undo stack which is discarded when the script editor is closed).

## 6.6 Event Logging

Errors that occur during a build are normally logged to the Output pane or console window and the log file if enabled (unless the related option is disabled). Some errors might occur when the log file is not available or when building in silent mode and are written to the Windows Event Viewer (accessible from Computer Management or Administrative Tools) instead (if the related option is enabled). The

following error conditions will be logged to the Event Viewer:

- Errors initializing or building from the console application (or the GUI App when running in silent mode).
- Errors expanding macros in the log filename, creating the path for the log file, or writing to the log file.

## 6.7 File Inclusion/Exclusion Matching

Many Visual Build file actions support fields for including and/or excluding the files to process. Visual Build actions support the following method for matching of filenames.

If the Include and Exclude fields are empty, all files in the specified folder (and subfolders if processing recursively) will match. Otherwise, the Include and Exclude fields can contain one or more file or folder names or wildcards/masks, each on a separate line. Only files matching all include values and not matching any exclude values will be processed. Matching is non-case-sensitive.

Three types of matching are supported:

### **#1 Broad Filename Matching**

To match a specific filename or filename wildcard in any matching folder, enter the filename or mask on a separate line.

Supported **wildcard** characters are \* (matches zero or more characters), ? (matches any single character), and . (matches the separator between the filename and extension).

For example:

ReadMe.txt Matches all files or folder containing files (at any level if recursive) named ReadMe.txt

\*.txt Matches all files or folder containing files (at any level if recursive) with a txt extension

Test\*. Matches all files or folder containing files (at any level if recursive) starting with Test and having no extension

VisBuildPro. Matches all files or folders containing files (at any level if recursive) beginning with

\* VisBuildPro and ending with any extension

W?x.t?t Matches any filename or folder containing files (at any level if recursive) starting with W, followed by any character, followed by x, and with extension of t, any character, followed by x

### **#2 Broad Folder Matching**

To match a specific folder or folder wildcard at any level, enter the folder name or mask on a separate line, prefixed with a slash / or backslash \ character. For example:

/Test Matches a folder (at any level if recursive) named Test

/Vis\* Matches a folder (at any level if recursive) starting with Vis

/\*Build Matches a folder (at any level if recursive) ending with Build

*Notes:*

- Inclusion/Exclusion of folders takes precedence over files -- if a folder is excluded, none of the files in a matching folder will be included, and if a folder is included, only files within matching folders will be matched. Any folder name within nested folders will be matched for include or exclude (for instance, all files in the subfolder `abc/xyz` would be excluded if `/abc` or `/xyz` was found in the exclusions).

### **#3 Advanced File Matching**

For *advanced* file matching, the include/exclude fields support extended file globbing pattern matching. This is specified by entering one or more folder names or wildcards separated by a slash / or

backslash \ character and ending with a filename or wildcard.

The folder names are matched starting with the immediate folder under the root folder specified for the action. An additional wildcard of \*\* can be specified to recursively match any folder name.

Some examples:

**/VisBuildPro/*.bld	Matches all .bld files within a folder named VisBuildPro, recursively at any level
Production/Files/ReadMe.txt	Matches the ReadMe.txt file only in the subfolder Production/Files
**/B**/Read*.*	Matches any file starting with Read in any subfolder under a folder starting with B at any level
B*/*.rc	Matches all files with .rc extension in any folder starting with a B only at the root level

## 6.8 Predefined Macros

### 6.8.1 System Macros

Visual Build provides the following predefined system macros and tools. Windows also defines some of these macros as environment variables, which will take precedence over Visual Build's definition (if enabled on the General application options page), and you can override these values by defining your own project or global macros or environment variables. Some values vary depending on the current date/time, or step being processed:

**COMPUTERNAME:** The current computer name.

**COMSPEC:** Path and filename of the command shell executable (cmd.exe).

*Note:* To call the 64-bit version of cmd.exe from the 32-bit edition of Visual Build, create a COMSPEC global macro with a value of %WINDIR%\Sysnative\cmd.exe; to call the 32-bit version of cmd.exe from the 64-bit edition of Visual Build, create a COMSPEC global macro with a value of %WINDIR%\SysWOW64\cmd.exe.

**DATE:** Inserts the current date. The date will be formatted according to the Short Date style in the Control Panel Regional Settings for dates.

**DATETIME:** Inserts the current date and time. The value will be formatted according to the Short Date and Time styles in the Control Panel Regional Settings for dates.

*Note:* Script code can also be used to insert date values with specific formatting. For instance, to insert the date in the format YYYY-MM-DD, use the VBScript expression [Year(Now) & "-" & Month(Now) & "-" & Day(Now)] in any step field. See the scripting [function reference on MSDN](#) for more details. System scripts for formatting dates are also provided.

**DOSCMD:** Executes an operating system command (i.e., DIR, COPY, ECHO, etc.), batch file, or command script using cmd.exe. Use this in a Run Program action to execute batch files, command scripts, individual shell commands, or to execute a program under the command shell for redirection (> or >>) or piping (|) of that program's output.

*Note:* To call the 64-bit cmd.exe from the 32-bit edition of Visual Build, create a COMSPEC global macro with a value of %WINDIR%\Sysnative\cmd.exe. To call the 32-bit version of cmd.exe from the 64-bit edition of Visual Build, create a COMSPEC global macro with a value of %WINDIR%\SysWOW64\cmd.exe.

**FAILSTEP\_STATUS:** Holds the build status of the project step that failed, can be used in Failure steps



to log the actual failed step info.

**FAILSTEP\_OUTPUT:** Contains the output from the failed step.

*Note:* The FAILSTEP\_OUTPUT macro includes any output logged by actions that was not output due to the step's Disable logging option being checked or the global logging level being lower than the message's log level.

**FAILSTEP\_NAME:** Contains the name of the failed step (equivalent to the script expression [FailedStep.Name]).

**LASTSTEP\_STATUS:** Holds the build status of the last step that was processed.

**LASTSTEP\_OUTPUT:** Contains any output from the last step that was processed. This can be useful for parsing the output for error information in the following step or the step's vbld\_StepDone script event.

*Note:* The LASTSTEP\_OUTPUT macro includes output logged by actions that was not output due to the step's Disable logging option being checked or the global logging level being lower than the message's log level.

**LASTSTEP\_NAME:** Contains the name of the last step that was processed (equivalent to the script expression [LastStep.Name]).

**LOGFILE:** The filename of the log file (an empty string if logging is not enabled). This will be the setting configuration in Application Options; it can also be overridden by a global, project, or temporary LOGFILE macro.

**PROFILES\_FULLPATH:** The full path and filename of the current file for the Process Files action.

**PROJDIR:** The full path of the open Visual Build project file. Useful for accessing other files that will be located relative to the project, allowing projects to work correctly no matter which drive or root directory they are placed at on a particular machine.

**PROJFILE:** The full path and filename of the open project file.

**PROJROOT:** The root name of the project file without extension.

**QUOTE\_STR:** Ensures that a string has double quotes around it.

**READ\_INI:** A macro used to read a value from an INI file and use in a step field. The syntax for calling it is: %READ\_INI(filename, section, valuenam, [default])%. Filename must be the full drive+path+filename for the file. The Read INI action can also be used to read INI files.

**READ\_XML:** A macro to read an element or attribute from an XML file and use in a build step. The syntax for calling it is: %READ\_XML(filename, xpath[, default][, namespaces])%. xpath denotes an XPath expression to search for; the text of the first matching node is returned or a blank string if not found. namespaces specifies one or more namespaces aliases separated by spaces (i.e., xmlns:bk='book' xmlns:n='http://testuri'). The Read XML action can also be used to read XML files.

*Note:* This macro requires MSXML version 3.0 or later to be installed.

**REG\_READ:** A macro to read a value from the registry and use in a build step. The syntax for calling it is: %REG\_READ("hive\key\value", ["default"])% . If an empty value is provided for the default parameter and the value is not found, an error occurs. To read the default value of a key, use an ending backslash character (i.e.,

`%REG_READ("HKEY_CURRENT_USER\Software\Test\Key\" , )%`. To read value names containing backslashes, use a double-backslash to delimit the key/value (i.e., `%REG_READ("HKEY_CURRENT_USER\Software\Test\\Value\Name" , )%` would read a value named *ValueName*). See the `RegEdit.bld` sample for more details.

*Note:* The `REG_READ` macro reads only 32-bit registry values on 64-bit windows. The Read Registry action can be used to read 32- and 64-bit registry values.

**TEMP:** The temporary path.

**TOOLS\_DIR:** The location of built-in Visual Build and miscellaneous third-party tools (usually `C:\Program Files\VisBuildPro9\Tools`). *Note:* This directory is only created for a Full installation.

**TRIM\_QUOTES:** Strips any double quotes surrounding a string value.

**USERNAME:** The logged on user's username.

**VISBUILD:** The full path to the current instance of Visual Build. Use to build another project from the current project (see the `Chain.bld` sample).

**VISBUILD\_CONFIG\_DIR:** The path to load Visual Build application and action data from.

**VISBUILDDIR:** The path of the current instance of Visual Build.

**VISBUILDVER:** The version of the current instance of Visual Build.

**WINDIR:** The Windows directory.

**WINSYSDIR:** The Windows System directory. *Note:* When running 32-bit edition of Visual Build on 64-bit Windows, Windows redirects this path to `%WINDIR%\SysWOW64`.

## 6.8.2 Global Macros

Visual Build creates the several global macros when first installed. By default, they are loaded/saved from/to the `VisBuildPro.macros` file in the configuration files path. Modifications to global macros are saved to disk whenever a project is saved, the GUI application is closed, and before each step is built. The values of these global macros will be initialized when Visual Build first run, but their values can be modified if necessary. Also, by clicking the *Reload* button on the File Locations options page, these values will be reinitialized to their default values.

Global macros are defined for several **Microsoft development tools**. Visual Build will attempt to locate the actual location of the tools and set the macros accordingly, but the value of these macros can be modified if necessary. Also, by clicking the *Reload* button on the File Locations property page, these values will be reinitialized to their default values. Note that these macros are *not* used by the built-in Microsoft actions.

**Embarcadero/CodeGear/Borland Products:** See the `Make Delphi/C++Builder` and `Make Delphi Prism` actions and `Embarcadero.bld` sample.

*Notes:* Global macros can be used for two-way communication between steps and projects. When a global macro is modified, in the GUI via the Macros pane, within a Set Macro or Run Script action, or from an external step (for instance, a child project via `VisBuildPro` Project action or an external script that instantiates and modifies a macro via the object model), only the in-memory macro collection is modified. However, when the objects holding that collection are disposed of (when the console or GUI app exits or references to the object model are released), if any global macro changes have been made, they will be saved to disk. Also, within a project, before each step is built, any in-memory

modifications to global macros will be saved, and if the global macros file gets modified by a step, the changes will be reloaded from disk when the step completes.

## 6.9 Regular Expressions

Many Visual Build actions support the use of [regular expressions](#) for matching and/or replacing text. Visual Build actions utilize the [Boost regex](#) and Microsoft [.NET](#) regex engines, which use a [syntax](#) compatible with [Perl regex](#).

*Note:* Bracket characters [ and ] and the percent sign character % within step fields have special meaning in Visual Build.

### Tutorials/References/Tools

- [Regular Expression Tutorial](#)
- [Regexlib.com](#)
- [RegexBuddy](#)
- [Mastering Regular Expressions](#)
- Regular Expressions Cookbook
- Microsoft [Regex Documentation](#)

### Overview

Description	Description	Expression	Matches
Literal Characters	Any non-special characters will match exactly what is typed	the dog	the dog
Special Characters	Meta-characters that have a special meaning in expressions: \ ^ \$ .   ? * + [ ] ( ) Use \ to escape a special character and match it	\\abc\\.	\\abc.
Any Character	Matches any character (except newline depending on configuration)	.	any single character
Character Class	Matches only one of several characters, or matches any character not in the class if negated	[[aeiou]] [[^y]] \\w \\d \\s \\W \\D \\S	any vowel character any character except y word character digit whitespace non-word character non-digit character non-white space character
Anchors	Match a position before, after, or between characters	^ \$ \\A \\z \\b \\B	beginning of line end of line beginning of string/file end of string/file word boundary not-word boundary
Repetition	Matches 0, 1, or more times	Jan(uary)? a* b+ x{1,3}	Jan or January nothing, a, aa, aaa, etc. b, bb, bbb, etc. x, xx, or xxx
Alternation	Match a single expression of several	cat dog bird	any of 'cat', 'dog',

Grouping	possible expressions Group part of an expression together	( (?: (?>	or 'bird' capture to numbered expression group without capture atomic grouping positive lookahead negative lookahead positive lookbehind negative lookbehind first matching group
Lookaround	Match characters, but give up the match and only return the result.	(?= (?! (?<= (?<!	
Back references	Reuse part of an expression or reference for substitution	\1 or \$1	

## 6.10 Special Characters

Some characters within macro values and step fields have special meaning in Visual Build:

- The percent sign character % is used around a macro name to indicate that macro's value should be expanded when the field is used (i.e., %DATETIME%). To insert a *literal* percent character, type two percent characters %%
- Bracket characters [ and ] denote a script expression to be inserted into a field (i.e., [vbld\_GetFileContents("c:\boot.ini")]). To insert *literal* bracket characters into a field, type two bracket characters [ [ or ] ].

### Notes:

- The vbld\_EscapeString system script function can be used to escape special characters in strings.
- The order of processing is: First, macro references are expanded (recursively); then script expressions are evaluated, and this process is repeated until no macro references or script expressions remain.
- To copy the current field's expanded value (with all macros expanded) to the clipboard, type Ctrl+M. To copy with macros expanded and with script evaluated, type Ctrl+Shift+M.
- When copying and pasting text from/to step properties and macro values to/from other applications, escaped special characters can automatically be unescaped (undoubled) when copying and escaped (doubled) when pasting if the related option is enabled. This can be useful, for instance, when copying regular expressions containing many bracket characters from another application.

## 6.11 Long Filenames

Many of the file processing actions in Visual Build call Windows APIs to do their work. Normally, these APIs are limited to paths up to 248 characters and filenames up to 260 characters, but they support paths and filenames up to 32,767 characters in length (extended-length paths) if the path or filename is prefixed with \\?\ (i.e., \\?\C:\Windows or \\?\UNC\Server\Share\path). The file processing actions in Visual Build automatically add this prefix to extended-length paths and filenames when calling Windows file and path APIs.

## 6.12 Usage Tips

- Check out the sample projects that are installed with Visual Build. A shortcut to the samples folder is available in the Visual Build group in the Start menu.
- See the Visual Build [FAQ](#) for additional samples and answers to common questions.

- Bracket characters [ and ] within a field normally denote a script expression to be inserted into a field (i.e., [ vbld\_GetFileContents("filename") ]); to insert literal brackets, use two bracket characters [ [ or ] ].
- The percent sign character % within a field normally denotes a macro to be expanded within a field (i.e., %PROJFILE%); to insert a literal percent character, use two percent characters %%
- To prompt the user for a value during the build (for instance to provide a unique build number), there are several methods that can be used:
  - 1) Reference an undefined macro in a project step (i.e., %MY\_PROMPT%). When that step is built, a dialog will be displayed prompting the user for the macro value (applies only to the GUI App). After the value has been entered, a temporary macro will be created with that value, and that value will be substituted in all steps that reference it (the temporary macro value will be discarded when the Visual Build instance is closed or rebuilt). One drawback to this method is that the user needs to know what an appropriate value for the macro should be.
  - 2) To display a custom message when prompting, create a Run Script action which uses the InputBox function to prompt for a value, and store the value in a temporary macro.
  - 3) If you want to provide even greater control over the prompted values (for instance, to prompt for many inputs or provide custom drop-down choices), you can create a custom user action (in the language of your choice) which prompts for all values and stores the user inputs in temporary macros. All three methods are demonstrated in the Prompt.bld sample.
- Group your project steps by different indentation levels to improve organization and to take advantage of relative Start In paths. Clicking the check box for a step quickly checks or unchecks all child steps as well. Clicking the folder icon of the top level *Project steps* item checks or unchecks all steps in the project.
- To force an individual step with a status of *Completed* or *Skipped* to be built again, rebuild the step or uncheck (this clears its status) and check the step again.
- Wrap Run Program action command filenames and parameters in double quotes (") if they contain spaces so that they are treated as a single argument.
- Any internal or external OS command, batch file, or command script can be called prefixing a Run Program command with the %DOSCMD% call system macro. This includes COPY, XCOPY, MKDIR, RMDIR, MOVE, ATTRIB, FIND, DEL, DIR, FORMAT, SYS, etc., plus any batch/CMD language statements (FOR, TYPE, ECHO, NET, etc.) or files. The output will be captured and displayed in the Output pane and log file if logging is enabled.
- To retrieve the contents of a file and use it within a step of the build, it can be returned via the system script function [ vbld\_GetFileContents("filename") ].
- To write the contents of a file to the build log, use a Run Program action with a command of %DOSCMD% TYPE "<filename>". or a Log Message action with a message of [ vbld\_GetFileContents("filename") ].
- Several methods are available for modularizing builds. First, subroutines can be defined and called from a project (subroutines can call other subroutines). Global subroutines can be used for functionality that is needed across multiple projects. And chaining can be used to invoke other Visual Build projects using the VisBuildPro Project action.
- There are many ways to pass dynamic values to or communicate between multiple projects:
  1. Temporary macro values can be passed on the command-line.
  2. The global macros collection can be used to communicate in both directions. Any changes to global macros are saved before each step is built and reloaded after a step completes if they were

modified externally, and the global macros are automatically saved when the Application object is disposed of. Global macros can be updated using a Set Macro action or a Run Script action (using the code `Application.Macros(vbldGlobal).Add "MY_MACRO", "some value"`); the macros can be read by referencing the macro in a step (i.e., `%MY_MACRO%`) or via the script code `vbld_AllMacros.Item("MY_MACRO").Value`.

3. Values can be read and written from the registry via the Write Registry and Read Registry actions.
  4. Values can be read and written from a file via the Write INI and Read INI actions.
  5. Values can be read and written from an XML file (see the XML.bld sample).
- Define a default value for a macro in a project macro (for instance, to use in a conditional build rule to determine which parts of a project are built), and then override that macro via a temporary macro (by passing as a parameter in a VisBuildPro Project action) in the situations where the default need to be overridden.
  - Several methods are available use a different log file for each project:
    1. Enter the filename on the Logging tab of the Project Properties dialog.
    2. Define a LOGFILE project macro.
    3. To use project-specific log files for every project, use a filename of `%PROJDIR%%PROJROOT%.<ext>` on the Logging tab of the Application Options dialog.
  - To apply the same conditional build rule to multiple steps, add a Group step that defines the build rule, then indent all the conditional steps below this one.

## 7 Object Model Reference

The Visual Build core server components are implemented as 32-bit and 64-bit COM (Component Object Model) classes and can be invoked from script (both internal and external to Visual Build) and from any other language that integrates with COM. All classes and interfaces support early and late (IDispatch) binding and use only automation-compatible types. This is a powerful capability that allows this and more:

1. Access the object model from a Run Script action, script events, or other step or macro properties within a build to access functionality that isn't exposed by the GUI app. The global items Application, Project, Step, LastStep, FailedStep, Builder, and WScript are available to all script code. For instance, this could be used to call script code within a conditional build rule to evaluate arbitrary expressions, to dynamically create a project and build it, and more.
2. Create a custom front-end to Visual Build that is specific to your projects. For instance, this could be used to provide a custom GUI to configure the build and launch it.
3. Access Visual Build from within your application or other environments (for instance, within Windows and .NET applications, Microsoft Excel, Word, etc.).

The COM interfaces and coclasses are defined in the type library found in `VisBuildSvr.dll` in the installation directory. The type library name is *Visual Build Professional 9 Server Objects*. There are two coclasses exposed in the type library: the *Application* class and the *Builder* class. Both of these classes are available as named items when creating script within Visual Build and are also publicly createable. The following named items are available from script code in script code, events, script actions, and the Run Script action:

Application  
Project  
Builder  
Step (the step being built)  
LastStep (the last step that was built)  
FailedStep (the last step that failed to build)  
WScript

The object model hierarchy is shown below:

```

Application
  Options
  Project
    Steps
      Step
    Macros
      Macro
  Macros
    Macro
  Scripts
    Script

```

```

Builder
  Application

```

```

WScript

```

In addition, all the custom action screens and components and logging components are implemented as COM plug-ins. See the user actions topic for information on creating your own custom action components, and see the VB logging sample component for an example of a custom logger.

## 7.1 Application Object

Holds all project and configuration data for the Visual Build application and the current project. The global Application item is available to all script code

**Progid:** `VisBuildSvr9.Application` or `VisBuildSvr.Application` (version-independent)

*Note:* The Application component is an in-process server marked for threading of Both (STA or MTA). If created from a GUI thread, any event handling in the calling code **must** marshal all events back to the GUI thread.

### Example

This Visual Basic 6.0 code demonstrates how to create an Application object and load a project:

```

Dim app As VisBuildSvr.Application
Set app = CreateObject("VisBuildSvr.Application")
app.Project.Load "c:\test.bld"
Debug.Print "Project Steps: " & app.Project.Steps(vbldStepMain).Count
Debug.Print "Global Macros: " & app.Project.Macros(vbldMacroGlobal).Count

```

### Properties

Context | Options | Project | Macros | Scripts | Steps

### Methods

FindMacro | ExpandMacros | ExpandMacrosAndScript | Initialize | Uninitialize

### Events

MacroModified | ProjectModified | ProjectSaved | PromptMacroValue

### See Also

**Builder** object

### 7.1.1 Context Property

Returns the context the build components were created from (read-only).

#### Syntax

*Application.Context*

#### Arguments

*Application*

**Application** object

*returns*

Context identifying how the application was created. Possible values:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vblDContextAuto</b>	0	The builder was created via automation (default).
<b>vblDContextGUI</b>	1	The builder was created by the GUI App.
<b>vblDContextCom</b>	2	The builder was created by the console app.

**mandLine**

#### See Also

Applies to Application object

### 7.1.2 Options Property

Provides the **IOptions** interface of the **Options** object.

*Application.Options As IOptions*

#### Arguments

*Application*

**Application** object

#### See Also

Applies to Application object

### 7.1.3 Project Property

Provides the **IProject** interface of the **Project** object.

*Application.Project As IProject*

#### Arguments

*Application*

**Application** object



**See Also**

Applies to Application object

**7.1.4 Macros Property**

Provides the **IMacros** interface of the **Macros** collection.

**Syntax**

```
Application.Macros(ByVal type As MacroTypeEnum) As IMacros
```

**Arguments**

*Application*

**Application** object

*type*

Required. The macro collection to retrieve. The values for *type* are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldMacroAll</b>	-1	A collection of all unique macros, with highest precedence for macros with same name in multiple collections.
<b>vbldMacroTemporary</b>	0	All temporary macros (not persisted between session; passed in on the command-line or created or loaded during a build)
<b>vbldMacroProject</b>	1	All project macros (saved with project file).
<b>vbldMacroGlobal</b>	2	All global macros. By default, global macros are loaded/saved from/to <code>VisBuildPro.macros</code> in the configuration files path.
<b>vbldMacroSystem</b>	3	All system macros (initialized internally; read-only)

**See Also**

Applies to Application object

**7.1.5 Scripts Property**

Provides the **IScripts** interface of the **Scripts** collection.

**Syntax**

```
Application.Scripts(ByVal type As ScriptTypeEnum) As IScripts
```

**Arguments**

*Application*

**Application** object

*type*

Required. The script collection to retrieve. The values for *type* are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
-----------------	--------------	--------------------

<b>vbldScriptAll</b>	-1	A collection of all script of the types below, concatenated together.
<b>vbldScriptTemporary</b>	0	Temporary scripts (not persisted between sessions; created or loaded during a build).
<b>vbldScriptProject</b>	1	Project scripts (saved with the project).
<b>vbldScriptGlobal</b>	2	Global scripts. By default, global scripts are loaded/saved from/to <code>VisBuildPro.scripts</code> in the configuration files path.
<b>vbldScriptSystem</b>	3	System scripts (stored in <code>VisBuildPro.System.scripts</code> ; read-only).

### See Also

Applies to Application object

## 7.1.6 Steps Property

Provides the **ISteps** interface of the global subroutine **Steps** collection.

### Syntax

```
Application.Steps As ISteps
```

### Arguments

*Application*  
**Application** object

### See Also

Applies to Application object

## 7.1.7 ExpandMacros Function

Expands macro references in the provided string, leaving undefined macros unchanged and evaluating script expressions only if the Shift key is held down.

### Syntax

```
Application.ExpandMacros(ByVal input As String) As String
```

### Arguments

*Application*  
**Application** object

*input*

Required. The string to search for and expand macro values in. If no macros are found, the string is returned unchanged.

### See Also

Applies to Application object

### 7.1.8 ExpandMacrosAndScript Function

Expands any macro references and script expressions in the provided string, treating undefined macros or script errors as an error condition.

#### Syntax

```
Application.ExpandMacrosAndScript(ByVal input As String) As String
```

#### Arguments

*Application*

**Application** object

*input*

Required. The string to search for and expand macro and script in. If no macros or script are found, the string is returned unchanged.

#### See Also

Applies to Application object

### 7.1.9 FindMacro Function

Search all macro collections for the given macro, searching highest precedence macros first.

#### Syntax

```
Application.FindMacro(ByVal name As String) As IMacro
```

#### Arguments

*Application*

**Application** object

*name*

Required. The name of the macro to search for.

*Return Value*

Returns the Macro object if found or Nothing if not found.

#### See Also

Applies to Application object

### 7.1.10 GlobalSubroutineStepsLoaded Event

Notification that occurs when the global subroutine steps are loaded via the object model.

#### Syntax

```
Application_GlobalSubroutineStepsLoaded( )
```

#### Arguments

*Application*

**Application** object

### See Also

Applies to Application object

## 7.1.11 Initialize Method

Initializes the application object.

### Syntax

```
Application.Initialize(ByVal context As BuilderCreateContextEnum, Optional
ByVal builder As IBuilder = Nothing)
```

### Arguments

*Application*

**Application** object

*context*

Required. Context identifying how the application was created. Possible values:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vblDContextAuto</b>	0	The builder was created via automation (default).
<b>vblDContextGUI</b>	1	The builder was created by the GUI App.
<b>vblDContextCom</b>	2	The builder was created by the console app.

*builder*

Optional. Builder object to initialize with. *Note:* providing this value will create a circular dependency between the Builder and Application objects. You must call Uninitialize to break the circular dependency to avoid memory leaks.

### See Also

Applies to Application object

## 7.1.12 Uninitialize Method

Uninitializes the application object. Breaks the circular reference created by passing the Builder object to Initialize.

### Syntax

```
Application.Uninitialize()
```

### Arguments

*Application*

**Application** object

**See Also**

Applies to Application object

**7.1.13 ProjectModified Event**

Notification that occurs when a project step, macro, script, or comment has been added, modified, or deleted.

**Syntax**

```
Application.ProjectModified()
```

**Arguments**

*Application*  
**Application** object

**See Also**

Applies to Application object

**7.1.14 ProjectSaved Event**

Notification that occurs when a project is saved.

**Syntax**

```
Application.ProjectSaved()
```

**Arguments**

*Application*  
**Application** object

**See Also**

Applies to Application object

**7.1.15 PromptMacroValue Event**

Notification that occurs when an undefined macro is encountered while expanding macros. The listener can prompt the user for a value.

**Syntax**

```
Application.PromptMacroValue(ByVal name As String, canceled As Boolean,  
value As String)
```

**Arguments**

*Application*  
**Application** object

*name*

The name of the undefined macro.

*canceled*

Callee must set to False if a value was provided by the user.

*value*

The value entered by the user.

**See Also**

Applies to Application object

**7.1.16 MacroModified Event**

Notification that occurs when a macro is being added, modified, or deleted.

**Syntax**

```
Application_MacroModified(ByVal macro As IMacro, ByVal operation As
ModificationTypeEnum)
```

**Arguments***Application*

**Application** object

*macro*

The macro that was modified or Null if multiple macros were affected.

*operation*

The modify operation that occurred. The values for *operation* are:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vbldModAdd</b>	0	The macro was added.
<b>vbldModUpd</b>	1	The macro was modified.
<b>ate</b>		
<b>vbldModDele2</b>		The macro is begin deleted.
<b>te</b>		

**See Also**

Applies to Application object

**7.2 Builder Object**

Performs a build of a Visual Build project. The global Builder item is available to all script code

**ProgId:** `VisBuildSvr9.Builder` or `VisBuildSvr.Builder` (version-independent)

*Note:* The Builder component is an in-process server marked for Free threading (MTA). A version marked for Both threading (MTA or STA) is also available.

**Properties**

App | BuildIndex | BuildThread | CallStack2 | CancelEvent | CompletionStatus | Debugging | FailedStep | LastStep | LaunchType | MaxStopWait | ProcessID | StartTime | Status | StepType

## Methods

ExpandMacros | GetFileContents | Initialize | LogFileContents | LogMessage | LogMessage2 | LogMessageEx | Pause | RegisterKey | Resume | RunProgram | RunProgram2 | RunProgramEx | RunProgramEx2 | ResetBuildStatus | Start | StartEx | StartEx2 | Stop | SyncBuild | SyncBuildEx | Uninitialize

## Events

BuildStarting | BuildDone | StepStarting | StepStarted | StepDone | BuildMessage

## Examples

*Note:* See the Object Model and Script.bld samples for additional examples.

This VBScript sample code demonstrates loading a project and building it:

```
Set objBld = CreateObject("VisBuildSvr.Builder")
Set objApp = CreateObject("VisBuildSvr.Application")
objBld.Initialize objApp

objApp.Project.Load "C:\Program Files\VisBuildPro9\Samples\RegEdit.bld"
objBld.SyncBuild
objBld.Uninitialize
```

This Visual Basic 6.0 sample code demonstrates loading a project, starting a build, and notifying the user when it completes:

```
Private WithEvents Builder As VisBuildSvr.Builder
Private m_app As VisBuildSvr.Application

Private Sub Builder_BuildDone(ByVal Status As VisBuildSvr.
BuildCompletionStatusEnum)
    MsgBox "Build completed with status code " & Status
End Sub

Private Sub Builder_BuildStarting()
    Debug.Print "starting"
End Sub

Private Sub cmdStart_Click()
    m_app.Project.Load txtFilename
    Builder.Start ' start, continue and handle events
    Debug.Print "started"
End Sub

Private Sub cmdStop_Click()
    Builder.Stop
End Sub

Private Sub Form_Load()
    Set Builder = New VisBuildSvr.Builder
    Set m_app = New VisBuildSvr.Application
    Builder.Initialize m_app
End Sub
```

```
Private Sub Form_Unload()  
    If Not Builder Is Nothing Then Builder.Uninitialize  
End Sub
```

### See Also

Application object

## 7.2.1 App Property

Returns the **Application** object to use for the build. Read-only.

### Syntax

```
builder.Application As IApplication
```

### Arguments

*builder*  
**Builder** object

### See Also

Applies to Builder object

## 7.2.2 BuildIndex Property

Returns the 0-based index of the current build step. Read-only.

### Syntax

```
builder.BuildIndex As Long
```

### Arguments

*builder*  
**Builder** object

### See Also

Applies to Builder object

## 7.2.3 BuildThread Property

Returns a handle to the Win32 thread that the build is executing on. Read-only.

### Syntax

```
builder.BuildThread As Long
```

### Arguments

*builder*  
**Builder** object



## See Also

Applies to Builder object

### 7.2.4 CallStack2 Property

Returns the current build call stack (an array of strings in the form <step type>;<step index>).  
Read-only.

#### Syntax

```
builder.CallStack2 As Array
```

#### Arguments

*builder*  
**Builder** object

## See Also

Applies to Builder object

### 7.2.5 CancelEvent Property

Returns a handle to the Win32 event that will be set if the build is aborted. Read-only.

#### Syntax

```
builder.CancelEvent As Long
```

#### Arguments

*builder*  
**Builder** object

## See Also

Applies to Builder object

A custom user action can use this event to test for user cancellation of the build. The VB6 code to perform this check is:

```
' in declares section
Private Declare Function WaitForSingleObject Lib "kernel32" Alias
"WaitForSingleObject" (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As Long

' in action's ICustomAction_BuildStep method
' should be called regularly during long a running action
' to check for cancellation of the build
If WaitForSingleObject(Builder.CancelEvent, 0) = 0 Then
    ICustomAction_BuildStep = vbldStepStatAborted
Exit Function
End If
```

## 7.2.6 CompletionStatus Property

Returns the completion status of the last build. Read-only.

### Syntax

```
builder.CompletionStatus As BuildCompletionStatusEnum
```

### Arguments

*builder*  
**Builder** object

### Values

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldBuildCompDone</b>	0	The build successfully completed.
<b>vbldBuildCompFailed</b>	1	A step failed to build.
<b>vbldBuildCompAborted</b>	2	The build was canceled by the user.

### See Also

Applies to Builder object

## 7.2.7 Debugging Property

Indicates if the current step is being debugged (single-stepped). Read-only.

### Syntax

```
builder.Debugging As Boolean
```

### Arguments

*builder*  
**Builder** object

### See Also

Applies to Builder object

## 7.2.8 FailedStep Property

Returns the failed step object.

### Syntax

```
Builder.FailedStep As IStep
```

### Arguments

*Builder*  
**Builder** object

*Return Value*

Returns the failed Step object or Nothing if none.

**See Also**

Applies to **Builder** object

**7.2.9 LastStep Property**

Returns the last built step object.

**Syntax**

```
Builder.LastStep As IStep
```

**Arguments***Builder*

**Builder** object

*Return Value*

Returns the last built Step object.

**See Also**

Applies to **Builder** object

**7.2.10 LaunchType Property**

Returns the launch type of the build. Useful for determining from within a step if the build was launched for build or rebuild. Read-only.

**Syntax**

```
builder.LaunchType As BuildLaunchTypeEnum
```

**Arguments***builder*

**Builder** object

*Values*

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vbldLaunchBuild</b>	0	Launched via Build, Build Group, or Build from Cursor.
<b>vbldLaunchRebuild</b>	1	Launched via Rebuild.
<b>vbldLaunchRebuildSel</b>	2	Launched via Rebuild Selected.
<b>vbldLaunchNone</b>	3	Other launch type (programmatically).

**See Also**

Applies to Builder object

### 7.2.11 MaxStopWait Property

Sets or returns the number of milliseconds to wait for a custom action to abort before forcibly terminating. Defaults to 5000 milliseconds.

#### Syntax

```
builder.MaxStopWait As Long
```

#### Arguments

*builder*  
**Builder** object

#### See Also

Applies to Builder object

### 7.2.12 ProcessID Property

Returns the process ID that the builder component is running in. Read-only. Used by the Set Priority action to set the build priority.

#### Syntax

```
builder.ProcessID As Long
```

#### Arguments

*builder*  
**Builder** object

#### See Also

Applies to Builder object

### 7.2.13 StartTime Property

Returns the date/time the current build was started. Read-only.

#### Syntax

```
builder.StartTime As Date
```

#### Arguments

*builder*  
**Builder** object

#### See Also

Applies to Builder object

## 7.2.14 Status Property

Returns the current status of the build. Read-only.

### Syntax

```
builder.Status As BuildStatusEnum
```

### Arguments

*builder*

**Builder** object

*Values*

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldBuildStatDone</b>	0	The build successfully completed or has not started.
<b>vbldBuildStatStarted</b>	1	The build is currently in progress.
<b>vbldBuildStatPauseReq</b>	2	The user has requested to pause the build.
<b>vbldBuildStatPaused</b>	3	The build has paused between steps.
<b>vbldBuildStatAborting</b>	4	The user has requested to cancel the build.

### See Also

Applies to Builder object

## 7.2.15 StepType Property

Returns the type of the currently building Step object. Read-only.

### Syntax

```
builder.StepType As StepTypeEnum
```

### Arguments

*builder*

**Builder** object

### See Also

Applies to Builder object

## 7.2.16 EvaluateRule method

Evaluates a step's build rule.

### Syntax

```
builder.EvaluateRule(ByVal step As IStep, Optional ByVal forDisplay As Boolean = False) As Integer
```

## Arguments

*builder*

**Builder** object

*step*

Required. The step object whose rule is to be evaluated.

*forDisplay*

Optional. If True, the results of rule evaluation are not logged and no prompting of undefined macro references will occur. If False, the rule evaluation will be logged and any undefined macro references will be prompted for.

*Return value*

The result of the step rule evaluation. The possible values are:

**Value** **Description**

0 The rule evaluates to 0 or False.

1 The rule evaluates to 1 or True.

-1 An error occurred evaluating the build rule.

## See Also

Applies to Builder object

## 7.2.17 ExpandMacros Method

Expands macros in a string, logging any errors to the build output.

### Syntax

```
builder.ExpandMacros(ByVal name As String, ByVal input As String, output As String, ByVal evalScript As Boolean = True) As Boolean
```

## Arguments

*builder*

**Builder** object

*name*

Required. The name of the property being expanded. This will be logged with any error information.

*input*

Required. The string to expand macros and script in.

*output*

Returned. The string with any macros or script expanded. The original string if no macros or script were found.

*evalScript*

Optional Whether to evaluate script in the expression surrounded by brackets.

*Return value*

True if the value was successfully expanded or False if errors occurred.

### See Also

Applies to Builder object

## 7.2.18 GetFileContents Method

Returns the contents of a file.

### Syntax

```
builder.GetFileContents(ByVal filename As String) As String
```

### Arguments

*builder*

**Builder** object

*filename*

Required. The filename to read.

### See Also

Applies to Builder object

## 7.2.19 Initialize Method

Initializes the builder for building a project, or reinitialize the build position. Must be called before building.

### Syntax

```
builder.Initialize(ByVal app As IApp = Nothing)
```

### Arguments

*builder*

**Builder** object

*app*

Optional. When first initialized, a valid Application object must be passed to identify the application and project to build. To reinitialize the build position and clear the call stack, it can be called without parameters.

### See Also

Applies to Builder object

## 7.2.20 LogFileContents Method

Logs the contents of a file to the build output.

### Syntax

```
builder.LogFileContents(ByVal filename As String)
```

### Arguments

*builder*

**Builder** object

*filename*

Required. The filename whose contents are to be logged.

### See Also

Applies to Builder object

## 7.2.21 LogMessage Method

Logs a message to the build output. The log level for messages logged with this function will have a level of Detailed. To specify another log level, use LogMessage2.

### Syntax

```
builder.LogMessage(ByVal message As String, ByVal appendCrLf As Boolean = True)
```

### Arguments

*builder*

**Builder** object

*message*

Required. The message to log.

*appendCrLf*

Optional. Whether to add a newline to the log output after the message.

### See Also

Applies to Builder object

## 7.2.22 LogMessage2 Method

Logs a message to the build output.

### Syntax

```
builder.LogMessage2(ByVal message As String, ByVal level As LogLevelEnum = vbldLogLevelDetailed, ByVal status As StepStatusEnum = vbldStepStatSucceeded, ByVal appendCrLf As Boolean = True) As StepStatusEnum
```

### Arguments

*builder*

**Builder** object

*message*

Required. The message to log.

*level*

Optional. The level of this message. The level specified and the globally configured logging level



will determine whether the message will be included in the console output, build log file, and Output pane. If the specified level is less than or equal to the configured log level, the message will be logged.

*status*

Optional. The status to return from the function.

*appendCrLf*

Optional. Whether to add a newline to the log output after the message.

### See Also

Applies to Builder object

## 7.2.23 LogMessageEx Method

Logs a message to the build output and returns the specified status.

### Syntax

```
builder.LogMessageEx(ByVal status As StepStatusEnum =  
vbldStepStatSucceeded, ByVal message As String, ByVal appendCrLf As Boolean  
= True) As StepStatusEnum
```

### Arguments

*builder*

**Builder** object

*message*

Required. The message to log.

*status*

Optional. The status to return from the function. If an error status is specified, a log level of Error will be used; otherwise a log level of Detailed will be used.

*appendCrLf*

Optional. Whether to add a newline to the log output after the message.

### See Also

Applies to Builder object

## 7.2.24 Pause Method

Requests the build to pause when the next step completes.

### Syntax

```
builder.Pause As Boolean
```

### Arguments

*builder*

**Builder** object

*Return value*

True if the build is in progress and the request has received, False if there was no current build.

### See Also

Applies to Builder object

## 7.2.25 RegisterKey Method

Registers Visual Build license information. Useful for unattended installation and registration of license information.

### Syntax

```
builder.RegisterKey(ByVal name As String, ByVal key As String)
```

### Arguments

*builder*

**Builder** object

*name*

The licensed name or company name and license count (required).

*key*

The license key for a purchased license (required).

### See Also

Applies to Builder object

## 7.2.26 ResetBuildStatus Method

Resets the build status of all steps in the project and reset build position to the start of the project.

### Syntax

```
builder.ResetBuildStatus
```

### Arguments

### See Also

Applies to Builder object

## 7.2.27 Resume Method

Requests the build to resume.

### Syntax

```
builder.Resume As Boolean
```

### Arguments

*builder*

**Builder** object

*Return value*

True if the build is paused and was resumed, False if the build was not paused.

### See Also

Applies to Builder object

## 7.2.28 RunProgram Method

Creates a process to run an external application, logging any output that the process generates. Any macros should be expanded before passing parameter values to the method.

*Notes:*

- This method cannot be called from script code because the *exitCode* parameter is required, passed by reference, and of type Long. Use *RunProgramEx* instead.
- Use *RunProgram2* from Run Program-derived custom actions with passwords in the command or to not use the extended wait property.

### Syntax

```
builder.RunProgram(ByVal command As String, ByVal startIn As String,  
exitCode As Long, ByVal outputLoc As OutputLocEnum = vbldOutputStdout,  
ByVal outputFile As String = "", ByVal delOutFile As Boolean = False,  
ByVal window As Boolean = False, ByVal wait As Boolean = True, ByVal  
successCodes As String = "0") As StepStatusEnum
```

### Arguments

*builder*

**Builder** object

*command*

Required. The command for the program to run.

*startIn*

Required. The path to set as the current directory for the launched process. Can be an empty string.

*exitCode*

Returned value. The exit code of the process.

*outputLoc*

Optional. The location to read output from. The values for *outputLoc* are:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vbldOutputNone</b>	0	Don't read any output from the program.
<b>vbldOutputStdout</b>	1	Read any output written to Stdout or Stderr.
<b>vbldOutputFile</b>	2	Read output from a file.

*outputFile*

Optional. The name of the output file to read if *outputLoc* is **vbldOutputFile**.

*delOutFile*

Optional. Whether to delete the output file before running the program if *outputLoc* is **vbldOutputFile**.

*window*

Optional. Whether to show a window for the process.

*wait*

Optional. Whether to wait for the process to complete or return as soon as the process is started.

*successCodes*

Optional. Exit codes to consider successful. In the form `code1, code2:code3`

*Return value*

The result of running the program. The possible values are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldStepStatSucceeded</b>	0	The step was successfully built or was marked to continue on failure.
<b>vbldStepStatFailed</b>	1	The step failed.
<b>vbldStepStatAborted</b>	2	The build was aborted by the user.

**See Also**

Applies to Builder object

**7.2.29 RunProgram2 Method**

Creates a process to run an external application, logging any output that the process generates.

*Notes:*

- This is an extended version of RunProgram with two additional parameters: *pwdCommand* and *useWaitProperty*.
- This method cannot be called from script code because the *exitCode* parameter is required, passed by reference, and of type Long. Use RunProgramEx2 instead.

**Syntax**

```
builder.RunProgram2(ByVal command As String, ByVal pwdCommand As String,
ByVal useWaitProperty As Boolean, ByVal startIn As String, exitCode As
Long, ByVal outputLoc As OutputLocEnum = vbldOutputStdout, ByVal outputFile
As String = "", ByVal delOutFile As Boolean = False, ByVal window As
Boolean = False, ByVal wait As Boolean = True, ByVal successCodes As String
= "0") As StepStatusEnum
```

**Arguments***pwdCommand*

Required. The command for the program to run with any passwords obscured (if provided, this is the command that will be logged if the *Log the command-line* option on the Advanced tab is checked).

*useWaitProperty*

Specifies whether the step's wait property (the *Wait for completion* option on the Advanced tab)

will be used.

### See Also

Applies to Builder object

See RunProgram method for other parameters and additional details

## 7.2.30 RunProgramEx Method

Runs an external application, with advanced options. Any macros should be expanded before passing parameter values to the method.

Notes:

- If the *wait* parameter (or wait property for Run Program-derived actions) is True, the exit code of the process will be stored in the RUNPROGRAM\_EXITCODE temporary macro.
- If the *wait* parameter is False, the process ID of the launched process will be stored in the RUNPROGRAM\_PROCESSID temporary macro.
- The *window* option will be ignored for GUI applications if the Always show GUI applications option is checked.
- Use RunProgramEx2 from Run Program-derived custom actions with passwords in the command or to not use the extended wait property.

### Syntax

```
builder.RunProgramEx(ByVal command As String, ByVal startIn As String = "",
ByRef exitCode As Variant = Null, ByVal outputLoc As OutputLocEnum =
vbldOutputStdout, ByVal outputFile As String = "", ByVal delOutFile As
Boolean = False, ByVal window As Boolean = False, ByVal wait As Boolean =
True, ByVal successCodes As String = "0", ByVal redirInput As
RedirectInputEnum = vbldRedirectNone, ByVal redirInputVal As String = "",
ByVal logOutput As Boolean = True, ByVal returnOutput As Boolean = False,
ByRef output As Variant = Null) As StepStatusEnum
```

### Arguments

*builder*

**Builder** object

*command*

Required. The command for the program to run.

*startIn*

Optional. The path to set as the current directory for the launched process.

*exitCode*

Optional; returned value. The exit code of the process.

*outputLoc*

Optional. The location to read output from. The values for *outputLoc* are:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vbldOutputNo</b>	0	Don't read any output from the program.
<b>ne</b>		
<b>vbldOutputSt</b>	1	Read any output written to Stdout or Stderr.
<b>dout</b>		
<b>vbldOutputFil</b>	2	Read output from a file.
<b>e</b>		

*outputFile*

Optional. The name of the output file to read if *outputLoc* is **vbldOutputFile**.

*delOutFile*

Optional. Whether to delete the output file before running the program if *outputLoc* is **vbldOutputFile**.

*window*

Optional. Whether to show a window for the process.

*wait*

Optional. Whether to wait for the process to complete or return as soon as the process is started. For Run Program-derived actions, this parameter will be ignored and the value of the extended wait step property will be used directly.

*successCodes*

Optional. Exit codes to consider successful. In the form `code1, code2:code3`

*redirectInput*

Optional. The location to provide redirected input from. The values for *redirectInput* are:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vbldRedirectN0</b>	one	Don't read send any input to the program.
<b>vbldRedirectF1</b>	file	Write the contents of the given file to the program's standard input.
<b>vbldRedirectS2</b>	tring	Write the given string to standard output.

*redirectInputVal*

Optional. The filename or string to provide program input from.

*logOutput*

Optional. Whether to log any captured output to the build log.

*returnOutput*

Optional. Whether to return the program's output in the output parameter.

*output*

Optional. A reference to a variable to populate with program output.

*Return value*

The result of running the program. The possible values are:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
<b>vbldStepStatSucceeded</b>	0	The step was successfully built or was marked to continue on failure.
<b>vbldStepStatFailed</b>	1	The step failed.
<b>vbldStepStatAborted</b>	2	The build was aborted by the user.

**See Also**

Applies to Builder object

### 7.2.31 RunProgramEx2 Method

Runs an external application, with advanced options. Any macros should be expanded before passing parameter values to the method.

*Notes:*

- If the wait parameter is True, the exit code of the process will be stored in the RUNPROGRAM\_EXITCODE temporary macro.
- If the wait parameter is False, the process ID of the launched process will be stored in the RUNPROGRAM\_PROCESSID temporary macro.

#### Syntax

```
builder.RunProgramEx2(ByVal command As String, ByVal startIn As String = "", , ByVal pwdCommand As String = "", ByVal useWaitProperty As Boolean = True, ByRef exitCode As Variant = Null, ByVal outputLoc As OutputLocEnum = vbldOutputStdout, ByVal outputFile As String = "", ByVal delOutFile As Boolean = False, ByVal window As Boolean = False, ByVal wait As Boolean = True, ByVal successCodes As String = "0", ByVal redirInput As RedirectInputEnum = vbldRedirectNone, ByVal redirInputVal As String = "", ByVal logOutput As Boolean = True, ByVal returnOutput As Boolean = False, ByRef output As Variant = Null) As StepStatusEnum
```

#### Arguments

*pwdCommand*

Required. The command for the program to run with any passwords obscured (if provided, this is the command that will be logged if the *Log the command-line* option on the Advanced tab is checked).

*useWaitProperty*

Specifies whether the step's wait property (the *Wait for completion* option on the Advanced tab) will be used.

#### See Also

Applies to Builder object

See RunProgramEx for other parameters and additional details

### 7.2.32 Start Method

Starts or continues a build.

#### Syntax

```
builder.Start(ByVal type As StepTypeEnum = -1, ByVal startIndex As Long = -1, ByVal count As Long = 0)
```

#### Arguments

*builder*

**Builder** object

*type*

Type of steps to start building. If -1 is passed, the build is started or continued from the current position.

*startIndex*

Index of step to start building. If -1 is passed, the build is started/continued from the current index.

*count*

Number of steps to build. If 0 is passed, all checked steps are evaluated for building.

**See Also**

Applies to Builder object

**7.2.33 StartEx Method**

Starts or continues a build, with extended options.

**Syntax**

```
builder.StartEx(ByVal launchType As BuildLaunchTypeEnum, ByVal stepType As StepTypeEnum = -1, ByVal startIndex As Long = -1, Optional ByVal selSteps As Variant)
```

**Arguments***builder*

**Builder** object

*launchType*

How the build is being launched.

*stepType*

Type of steps to start building. If -1 is passed, the build is started or continued from the current position.

*startIndex*

Index to set the current build position to. If -1 is passed, the build is started/continued from the current build position.

*selSteps*

Array of Long values containing one or more selection ranges (first and last step index) to be built. If not passed, all checked steps are evaluated for building.

**See Also**

Applies to Builder object

**7.2.34 StartEx2 Method**

Starts or continues a build, with extended options and logon credentials.

**Syntax**

```
builder.StartEx2(ByVal launchType As BuildLaunchTypeEnum, ByVal flowThreadIdentity As Boolean = False, ByVal processIdentityUserName = "", ByVal processIdentityDomain = "", ByVal processIdentityPassword = "",
```



```
ByVal logonNetCredentialsOnly As Boolean = True,  
useInteractiveWinStaDesktop As Boolean = True, ByVal stepType As  
StepTypeEnum = -1, ByVal startIndex As Long = -1, Optional ByVal selSteps  
As Variant)
```

## Arguments

### *builder*

**Builder** object

### *launchType*

How the build is being launched.

### *flowThreadIdentity*

Whether to use the impersonated identity of the calling thread for the build thread. If False, the default identity of the process will be used for the build thread.

### *processIdentityUserName*

The user account name for an identity to use for any process started by build actions or script that call RunProgram/Ex or a Run Program action. This is the name of the user account to log on to. If you use the UPN format, user@DNS\_domain\_name, the domain field must be blank. The user account must have the Log On Locally permission on the local computer. This permission is granted to all users on workstations and servers, but only to administrators on domain controllers. If specified, the process will be created with the specified credentials. If blank, the process will be created using the default identity of the calling process.

*Note:* When specifying user credentials, processes started by the build will inherit the environment block of the calling process.

### *processIdentityDomain*

The name of the domain or server whose account database contains the user account specified in processIdentityUserName. Can be blank or . for the current domain.

### *processIdentityPassword*

The password of the user account for an identity to use for any processes started by the build.

### *logonNetCredentialsOnly*

Applies only if credentials are specified in the previous parameters. If True, processes created by the build will use the specified credentials on the network only. The new process uses the same token as the caller, but the system creates a new logon session within LSA, and the process uses the specified credentials as the default credentials. This can be used to create a process that uses a different set of credentials locally than it does remotely. This is useful in inter-domain scenarios where there is no trust relationship. The system does not validate the specified credentials. Therefore, the process can start, but it may not have access to network resources. If False, new processes will load the specified user's profile in the HKEY\_USERS registry key. Loading the profile can be time-consuming, so it is best to pass False only if the process must access the information in the HKEY\_CURRENT\_USER registry key.

### *useInteractiveWinStaDesktop*

Applies only if credentials are specified in the previous parameters. If False, new processes created by the build will inherit the desktop and windowstation of the calling process (permission for the specified user account to the inherited windowstation and desktop are added automatically). If True, the interactive windowstation and desktop will be used by processes created by the build.

*Note:* Permissions for the specified user account are not automatically added to the interactive windowstation and desktop when True is passed, and applications may not start or draw properly

unless the necessary permissions are assigned manually (see <http://support.microsoft.com/kb/q165194/> for more details).

*stepType*

Type of steps to start building. If -1 is passed, the build is started or continued from the current position.

*startIndex*

Index to set the current build position to. If -1 is passed, the build is started/continued from the current build position.

*selSteps*

Array of Long values containing one or more selection ranges (first and last step index) to be built. If not passed, all checked steps are evaluated for building.

**See Also**

Applies to Builder object

### 7.2.35 Stop Method

Request the build to abort.

**Syntax**

*builder*.Stop As Boolean

**Arguments**

*builder*

**Builder** object

*Return value*

True if a build is in progress and the request has been initiated, False if there was no current build.

**See Also**

Applies to Builder object

### 7.2.36 SyncBuild Method

Performs a synchronous build, waiting for the build to complete before returning.

**Syntax**

*builder*.SyncBuild() As BuildCompletionStatus

**Arguments**

*builder*

**Builder** object

*Return value*

The result of the build.

## See Also

Applies to Builder object

### 7.2.37 SyncBuildEx Method

Performs a synchronous build with logon credentials, waiting for the build to complete before returning.

#### Syntax

```
builder.SyncBuildEx(ByVal flowThreadIdentity As Boolean = False, ByVal  
processIdentityUserName = "", ByVal processIdentityDomain = "", ByVal  
processIdentityPassword = "", ByVal logonNetCredentialsOnly As Boolean =  
True useInteractiveWinStaDesktop As Boolean = True) As  
BuildCompletionStatus
```

#### Arguments

*builder*

**Builder** object

*flowThreadIdentity*

Whether to use the impersonated identity of the calling thread for the build thread. If False, the default identity of the process will be used for the build thread.

*processIdentityUserName*

The user account name for a logon identity to use for any process started by build actions or script that call RunProgram/Ex or a Run Program or derived action. This is the name of the user account to log on to. If you use the UPN format, user@DNS\_domain\_name, the domain field must be blank. The user account must have the Log On Locally permission on the local computer. This permission is granted to all users on workstations and servers, but only to administrators on domain controllers. If specified, the process will be created with the specified credentials. If blank, the process will be created using the default identity of the calling process.

*Note:* When specifying user credentials, processes started by the build will inherit the environment block of the calling process.

*processIdentityDomain*

The name of the domain or server whose account database contains the user account specified in processIdentityUserName. Can be blank or . for the current domain.

*processIdentityPassword*

The password of the user account for a logon identity to use for any processes started by the build.

*logonNetCredentialsOnly*

Applies only if credentials are specified in the previous parameters. If True, processes created by the build will use the specified credentials on the network only. The new process uses the same token as the caller, but the system creates a new logon session within LSA, and the process uses the specified credentials as the default credentials. This can be used to create a process that uses a different set of credentials locally than it does remotely. This is useful in inter-domain scenarios where there is no trust relationship. The system does not validate the specified credentials. Therefore, the process can start, but it may not have access to network resources. If False, new processes will load the specified user's profile in the HKEY\_USERS registry key unless the UseProfileOptionWithSyncBuildEx option is set to False (in this case, an option value of 0 will be specified when creating processes). Loading the profile can be time-consuming, so it is best to pass False only if the process must access the information in the

HKEY\_CURRENT\_USER registry key.

*useInteractiveWinStaDesktop*

Applies only if credentials are specified in the previous parameters. If False, new processes created by the build will inherit the desktop and window station of the calling process (permission for the specified user account to the inherited window station and desktop are added automatically). If True, the interactive windowstation and desktop will be used by processes created by the build.

*Note:* Permissions for the specified user account are not automatically added to the interactive window station and desktop when True is passed, and GUI applications will not have full access to draw properly unless the necessary permissions are assigned manually (see here for more details).

*Return value*

The result of the build.

**See Also**

Applies to Builder object

### 7.2.38 Uninitialize Method

Uninitializes the application object. Breaks the circular reference created by passing the Application object to Initialize.

**Syntax**

*Builder*.Uninitialize()

**Arguments**

*None*

**See Also**

Applies to Builder object

### 7.2.39 BuildStarting Event

Occurs when a build begins. Custom script event code can be added for this event.

**Syntax**

*builder*\_BuildStarting()

**Arguments**

*builder*

**Builder** object

**See Also**

Applies to Builder object

### 7.2.40 BuildDone Event

Occurs when a build finishes. Custom script event code can be added for this event.

#### Syntax

```
builder_BuildDone(ByVal status As BuildCompletionStatus)
```

#### Arguments

*builder*

**Builder** object

*status*

The completion result of the build.

#### See Also

Applies to Builder object

### 7.2.41 StepStarting Event

Occurs when a step is about to be built. Custom script event code can be added for this event.

#### Syntax

```
builder_StepStarting(ByVal step As IStep, skip As Boolean)
```

#### Arguments

*builder*

**Builder** object

*step*

The step that is about to be built.

*skip*

Set this value to True if the step should be skipped.

#### See Also

Applies to Builder object

### 7.2.42 StepStarted Event

Occurs after a step has started building. Custom script event code can be added for this event.

#### Syntax

```
builder_StepStarted(ByVal step As IStep)
```

#### Arguments

*builder*

**Builder** object

*step*

The step that is about to be built.

### See Also

Applies to Builder object

## 7.2.43 StepDone Event

Occurs after a step has completed building. Custom script event code can be added for this event.

### Syntax

```
builder_StepDone(ByVal step As IStep)
```

### Arguments

*builder*

**Builder** object

*step*

The step that was built. The status of the step can be accessed via the step's BuildStatus property.

### See Also

Applies to Builder object

## 7.2.44 BuildMessage Event

Occurs for each build message that is logged (logging components catch this event to log all messages).

### Syntax

```
builder_BuildMessage(ByVal message As String, ByVal level As LogLevelEnum)
```

### Arguments

*builder*

**Builder** object

*level*

The level of the log message (determines whether it will be logged).

*message*

The message being logged.

### See Also

Applies to Builder object

## 7.3 Macro Object

Holds information for a Visual Build macro.

### Properties

Parent | Type | Category | Description | Name | **Value** | Parameters | AddAsEnvVar

### See Also

Macros collection

### 7.3.1 Category Property

Sets or returns the category of the macro.

#### Syntax

```
Macro.Category As String
```

#### Arguments

*Macro*  
**Macro** object

#### See Also

Applies to Macro object

### 7.3.2 Name Property

Sets or returns the name of the macro. Macro names cannot contain parentheses characters, percent signs, commas, or equal signs.

#### Syntax

```
Macro.Name As String
```

#### Arguments

*Macro*  
**Macro** object

#### See Also

Applies to Macro object

### 7.3.3 Value Property

Sets or returns the value of the macro. Any type of value, including numbers, strings, and objects, can be stored in macro values (objects should only be stored in temporary macros since they will not be persisted between sessions). Macros can reference other macros but cannot reference themselves.

#### Syntax

*Macro.Value As Variant*

### Arguments

*Macro*

**Macro** object

### See Also

Applies to Macro object

## 7.3.4 Parameters Property

Sets or returns the parameters of the macro. Entered as a comma-delimited list of parameter names. Parameter names cannot contain commas, percents, or parenthesis characters. All parameter names must be referenced in the macro value.

### Syntax

*Macro.Parameters As String*

### Arguments

*Macro*

**Macro** object

### See Also

Applies to Macro object

## 7.3.5 AddAsEnvVar Property

Sets or returns whether the macro will be added to the environment variables for any external programs launched from the build. *Note:* This property does not take effect from script code called in the current build. Use a Set Macro action instead (or see the Advanced.bld sample for a technique to dynamically generate a project with Set Macro steps marked to update environment variables and build it).

### Syntax

*Macro.AddAsEnvVar As Boolean*

### Arguments

*Macro*

**Macro** object

### See Also

Applies to Macro object



## 7.4 Macros Collection

Holds all macros of a given type.

### Properties

Parent | Count | IsModified | Name | Type | **Item** | \_NewEnum

### Methods

Add | AddEx | Remove | Clear | Load | Save

### See Also

Macros property

### 7.4.1 Type Property

Returns the type of macros in the collection. Read-only.

#### Syntax

```
object.Type As MacroTypeEnum
```

#### Arguments

*object*

An object expression that evaluates to an object in the Applies To list.

The return values are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbIdMacroAll</b>	-1	A collection of all unique macros, with highest precedence for macros with same name in multiple collections.
<b>vbIdMacroTemporary</b>	0	All temporary macros (not persisted between session; passed in on the command-line or created or loaded during a build)
<b>vbIdMacroProject</b>	1	All project macros (saved with project file).
<b>vbIdMacroGlobal</b>	2	All global macros. By default, global macros are loaded/saved from/to <code>VisBuildPro.macros</code> in the configuration files path.
<b>vbIdMacroSystem</b>	3	All system macros (initialized internally; read-only)

### See Also

Applies to Macros collection, Macro object

### 7.4.2 Item Property

Returns the given **Macro** object from the collection.

#### Syntax

```
macros.Item(ByVal name As String) As IMacro
```

## Arguments

*macros*

**Macros** collection

*name*

Required. Name of the macro. Names are compared without case sensitivity.

*Return Value*

Returns the Macro object if found or Nothing if not found.

## See Also

Applies to Macros collection

### 7.4.3 Name Property

The filename of the collection. Applies only to global macros, global scripts, global steps, and application options. Read-only.

## Syntax

```
Object.Name As String
```

## See Also

Applies to Macros collection, Scripts collection, Steps collection, Options object

### 7.4.4 Add Method

Adds or updates a **Macro** object in the collection.

## Syntax

```
macros.Add(ByVal name As String, ByVal value As Variant = Empty, ByVal  
params As String = "", ByVal descr As String = "", ByVal envVar As Boolean  
= False) As IMacro
```

## Arguments

*macros*

**Macros** collection

*name*

Required. Name of the macro. Names are compared without case sensitivity.

*value*

Optional. Value of the macro.

*params*

Optional. Parameters of the macro.

*descr*

Optional. Macro description.

*envVar*

Optional. Whether the macro will be added as an environment variable when building. *Note:* This property does not take effect from script code until the build is restarted. Use a Set Macro action (see the Script.bld sample for a technique to dynamically generate a project with Set Macro steps and build it).

*Return Value*

If the macro already exists, its properties are updated to the specified values.

**See Also**

Applies to Macros collection

### 7.4.5 AddEx Method

Adds or updates a **Macro** object in the collection.

**Syntax**

```
macros.AddEx(ByVal name As String, ByVal value As Variant = Empty, ByVal  
params As String = "", ByVal descr As String = "", ByVal envVar As Boolean  
= False, ByVal encrypt As Boolean = False) As IMacro
```

**Arguments***macros*

**Macros** collection

*name*

Required. Name of the macro. Names are compared without case sensitivity.

*value*

Optional. Value of the macro.

*params*

Optional. Parameters of the macro.

*descr*

Optional. Macro description.

*envVar*

Optional. Whether the macro will be added as an environment variable when building. *Note:* This property does not take effect from script code until the build is restarted. Use a Set Macro action (see the Script.bld sample for a technique to dynamically generate a project with Set Macro steps and build it).

*encrypt*

Optional. Whether the macro value will be stored encrypted in the project file.

*Return Value*

If the macro already exists, its properties are updated to the specified values.

**See Also**

Applies to Macros collection

### 7.4.6 Remove Method

Removes a **Macro** object from the collection.

*Note:* Any references to the macro should be disposed of (`Set objMacro = Nothing`) before removing.

#### Syntax

```
macros.Remove(ByVal name As String) As Boolean
```

#### Arguments

*macros*

**Macros** collection

*name*

Required. Name of the macro. Names are compared without case sensitivity.

*Return value*

True if the macro was found and removed or False if the macro was not found.

#### See Also

Applies to Macros collection

### 7.4.7 Load Method

Loads macros, steps, or script from a file.

#### Syntax

```
object.Load(ByVal filename As String, ByVal clear As Boolean = *)
```

#### Arguments

*object*

collection

*filename*

Required. The filename to load from.

*clear*

Optional. Whether to clear the collection before loading. If False, items are added to the existing collection (macros of the same name will overwrite existing macros). Defaults to False for macros and True for steps or script collections.

#### See Also

Applies to **Macros** collection, **Steps** collection, and **Scripts** collection

### 7.4.8 Save Method

Saves a macro, step, or script collection to a file.

## Syntax

```
object.Save(ByVal filename As String)
```

## Arguments

*object*  
collection

*filename*  
Required. The filename to save to.

## See Also

Applies to **Macros** collection, **Steps** collection, and **Scripts** collection

## 7.5 Options Object

Holds all configuration data for the Visual Build Builder object. At startup, any default options from `VisBuildPro.config` in the configuration files path are loaded.

### Properties

AlwaysShowGUIApps | BuildFailureStepsOnCancel | CacheActions | CaseSensitiveBuildRuleComparisons | CallStepStartingScriptEventOnSkippedStep | ConfigFilesPath | ConvertOutputDoubleQuotes | DefaultScriptEngine | DelLogFileOnBuild | DelTempMacrosAfterBuild | EchoChainedConsoleOutput | EnableEventErrorLogging | EncryptPasswordProperties | EnvVarsInSystemMacros | EscapeSpecialCharactersInOutput | FailBuildWhenDoneIfStepsFailed | HidePasswordPropertyValues | LogAllFailureStepOutput | LogDefaultStepProperty | LogFilename | LogFormat | LogLevel | MacroExpandBuildTimeout | MacroExpandTooltipTimeout | MaxDefaultPropertyLogLength | MaxMacroLengthExpand | MaxStepOutputLength | Name | NestBuildRules | NestIncludeInBuild | Parent | PersistBuildStatus | PreventDuplicateEventLog | PsExecCmd | ReevaluateAllBuildRules | RunProgramOutputBufferSize | RunProgramInputWaitTimeout | RunProgramOutputWaitTimeout | RunProgramRedirInputChunkSize | ScriptTypeLibraries | SetProjectDirectory | StripLogLinefeedChar | TextLogSkippedSteps | TextLogStepEvents | TextLoggerCacheMessages | TextLoggerCacheSeconds | UniqueEncryptionIV | UseLogonCreateProcessAsUser | UseProfileOptionWithSyncBuildEx | UseUSEnglishLocaleForScriptEngine | UseUTF8ForConsoleApps | WriteBOM | XMLLoggerCacheMessages | XMLLoggerCacheSeconds

### Methods

Load | Save

### See Also

Application object

### 7.5.1 AlwaysShowGUIApps Property

Determines whether a GUI application's window will be hidden when run from a Run Program or derived action or RunProgramEx call.

### Syntax

`Options.AlwaysShowGUIApps` As Boolean

#### **See Also**

Applies to Options object

### **7.5.2 BuildFailureStepsOnCancel Property**

Whether to build failure steps if the build is stopped.

#### **Syntax**

`Options.BuildFailureStepsOnCancel` As Boolean

#### **See Also**

Applies to Options object

### **7.5.3 CallStepStartingScriptEventOnSkippedStep Property**

Whether to call the StepStarting script events for a skipped step.

#### **Syntax**

`Options.CallStepStartingScriptEventOnSkippedStep` As Boolean

#### **See Also**

Applies to Options object

### **7.5.4 CacheActions Property**

Whether to cache action info and components when building.

#### **Syntax**

`Options.CacheActions` As Boolean

#### **See Also**

Applies to Options object

### **7.5.5 CaseSensitiveBuildRuleComparisons Property**

Whether to use case-sensitive string comparisons in conditional build rules.

#### **Syntax**

`Options.CaseSensitiveBuildRuleComparisons` As Boolean

#### **See Also**

Applies to Options object

### **7.5.6 ConfigFilesPath Property**

The path to load and save application data to.

**Syntax**

```
Options.ConfigFilePath As String
```

**See Also**

Applies to Options object

**7.5.7 ConvertOutputDoubleQuotes Property**

Whether to convert double quotes in the LASTSTEP\_OUTPUT and FAILSTEP\_OUTPUT system macros to single quotes. Enabling this can be useful if the macro will be referenced in script expressions, but it can cause problems for quoted values in XML output containing single quotes.

**Syntax**

```
Options.ConvertOutputDoubleQuotes As Boolean
```

**See Also**

Applies to Options object

**7.5.8 DefaultScriptEngine Property**

The ProgId of the default script engine, used when evaluating script expressions and executing script events.

**Syntax**

```
Options.DefaultScriptEngine As String
```

**See Also**

Applies to Options object

**7.5.9 DelLogFileOnBuild Property**

Whether to delete the log file at the beginning of a build.

**Syntax**

```
Options.DelLogFileOnBuild As Boolean
```

**See Also**

Applies to Options object

**7.5.10 DelTempMacrosAfterBuild Property**

Whether to delete temporary macros after build completes.

**Syntax**

```
Options.DelTempMacrosAfterBuild As Boolean
```

**See Also**

Applies to Options object

**7.5.11 EchoChainedConsoleOutput Property**

Whether to echo the output of console programs called from a build using the Console application. If True, the output of any console programs called by a Run Program or VisBuildPro Project step or any action that calls the Builder object's RunProgram or RunProgramEx methods will be echoed to the console that VisBuildCmd is running in.

**Syntax**

```
Options.EchoChainedConsoleOutput As Boolean
```

**See Also**

Applies to Options object

**7.5.12 EnableEventErrorLogging Property**

Whether to log severe build errors to the Windows Event Log.

**Syntax**

```
Options.EnableEventErrorLogging As Boolean
```

**See Also**

Applies to Options object

**7.5.14 EnvVarsInSystemMacros Property**

Whether to include environment variables in system macros.

**Syntax**

```
Options.EnvVarsInSystemMacros As Boolean
```

**See Also**

Applies to Options object

**7.5.15 EscapeSpecialCharactersInOutput Property**

Whether to escape characters in the LASTSTEP\_OUTPUT and FAILSTEP\_OUTPUT system macros. Within step and macro fields, bracket characters [ and ] normally denote a script expression to be inserted into a field, and the percent sign character % is normally used around a macro name for its value to be expanded within the field. If this option enabled, these special characters in step output will be doubled [ [ ] ] %%.

*Note:* This option should be enabled if the output macros will be referenced in step fields (i.e., %LASTSTEP\_OUTPUT%) so that any special characters will be treated as literal characters by Visual Build.

**Syntax**



`Options.ConvertOutputDoubleQuotes As Boolean`

### See Also

Applies to Options object

## 7.5.16 FailBuildWhenDoneIfStepsFailed Property

Whether to fail the overall build if any steps failed during the build (but the build continued).

### Syntax

`Options.FailBuildWhenDoneIfStepsFailed As Boolean`

### See Also

Applies to Options object

## 7.5.17 HidePasswordPropertyValues Property

Whether to hide (display asterisks for) encrypted step properties and macro values when displayed and logged in Visual Build.

### Syntax

`Options.HidePasswordPropertyValues As Boolean`

### See Also

Applies to Options object

## 7.5.18 LogAllFailureStepOutput Property

Determines whether a message with a higher level than the log level setting will be logged from a failure step .

### Syntax

`Options.LogAllFailureStepOutput As Boolean`

### See Also

Applies to Options object

## 7.5.19 LogDefaultStepProperty Property

Determines whether a step's default property is logged before building.

### Syntax

`Options.LogDefaultStepProperty As Boolean`

### See Also

Applies to Options object

### 7.5.20 LogBuildRuleEval Property

Determines whether a step's default property is logged before building.

#### Syntax

```
Options.LogDefaultStepProperty As Boolean
```

#### See Also

Applies to Options object

### 7.5.21 LogFilename Property

The filename of the log file to write to (also accessed as the LOGFILE system macro) or an empty string if logging is disabled.

#### Syntax

```
Options.LogFilename As String
```

#### See Also

Applies to Options object

### 7.5.22 LogFormat Property

The format to use when logging to a file. The supported formats are *XML* and *Text*.

#### Syntax

```
Options.LogFormat As String
```

#### See Also

Applies to Options object

### 7.5.23 LogLevel Property

Returns or sets the logging level for text file logging, build output, or text file logging or specifies the log level for a message when calling LogMessage, LogMessage2, and LogMessageEx.

#### Syntax

```
Options.LogLevel As LogLevelEnum
```

The values are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldLogLevelNone</b>	-1	No calls to LogMessage will be logged.
<b>vbldLogLevelError</b>	0	Only error level messages will be logged.
<b>vbldLogLevelWarning</b>	1	Only error and warning level error messages will be logged.
<b>vbldLogLevelNormal</b>	2	Errors, warnings, and normal level error messages will be logged.
<b>vbldLogLevelDetailed</b>	3	Errors, warnings, normal, and detailed error messages

**vbldLogLevelDiagnostic** 4 will be logged.  
Errors, warnings, normal, detailed, and diagnostic error messages will be logged.

### See Also

Applies to Options object

## 7.5.24 MacroExpandBuildTimeout Property

Timeout (in milliseconds) after which to terminate macro and script expansion when building. A value of 0 disables the timeout.

### Syntax

*Options*.MacroExpandBuildTimeout As Long

### See Also

Applies to Options object

## 7.5.25 MacroExpandTooltipTimeout Property

Timeout (in milliseconds) after which to terminate macro and script expansion for tool tips. A value of 0 disables the timeout.

### Syntax

*Options*.MacroExpandTooltipTimeout As Long

### See Also

Applies to Options object

## 7.5.26 MaxDefaultPropertyLogLength Property

Maximum length to log for a default step property. The property value will be truncated if it exceeds this length. A value of 0 sets no limit (doesn't truncate).

### Syntax

*Options*.MaxDefaultPropertyLogLength As Long

### See Also

Applies to Options object

## 7.5.27 MaxMacroLengthExpand Property

Maximum length of macro value to process for macro/script references. A value of 0 sets no limit.

### Syntax

*Options*.MaxMacroLengthExpand As Long

### See Also

Applies to Options object

### 7.5.28 MaxStepOutputLength Property

Maximum number of characters to capture from RunProgram / RunProgramEx output.

#### Syntax

```
Options.MaxStepOutputLength As Long
```

#### See Also

Applies to Options object

### 7.5.29 NestBuildRules Property

Whether to nest conditional build rules. If this option is True, nesting of build rules is performed: If a step and one or more of its parent steps contains a conditional build rule, all the rules must evaluate to true for that step to be built. Conditional build rules are not evaluated for steps that are not marked to be included in the build.

#### Syntax

```
Options.NestBuildRules As Boolean
```

#### See Also

Applies to Options object

### 7.5.30 NestIncludeInBuild Property

Whether to require the *include in build* step flag of all parent steps to be checked. If True, nesting of the checked flag is performed: A child step will only be considered for building if its parent step is checked. If not checked, the checked status of parent steps does not exclude the child step from the build. The *include in build* step flag is considered before evaluation of conditional build rules.

#### Syntax

```
Options.NestIncludeInBuild As Boolean
```

#### See Also

Applies to Options object

### 7.5.31 PersistBuildStatus Property

Whether to save each step's build status in the project file when saving.

#### Syntax

```
Options.PersistBuildStatus As Boolean
```

#### See Also

Applies to Options object

### 7.5.32 PreventDuplicateEventLog Property

Whether to prevent duplicate error messages in Event Viewer.

#### Syntax

```
Options.PreventDuplicateEventLog As Boolean
```

#### See Also

Applies to Options object

### 7.5.33 PsExecCmd Property

Specifies the command to execute for remote execution. Defaults to PsExec, but can be used to specify an alternate program (i.e., PAExec).

#### Syntax

```
Options.PsExecCmd As String
```

#### See Also

Applies to Options object

### 7.5.34 ReevaluateAllBuildRules Property

Whether to reevaluate all conditional build rules for a step before building.

#### Syntax

```
Options.ReevaluateAllBuildRules As Boolean
```

#### See Also

Applies to Options object

### 7.5.35 RunProgramInputWaitTimeout Property

Milliseconds to wait for more input from RunProgram / RunProgramEx output.

#### Syntax

```
Options.RunProgramInputWaitTimeout As Long
```

#### See Also

Applies to Options object

### 7.5.36 RunProgramOutputBufferSize Property

Size in bytes to buffer RunProgram / RunProgramEx output.

**Syntax**

`Options.RunProgramOutputBufferSize` As Long

**See Also**

Applies to Options object

**7.5.37 RunProgramOutputWaitTimeout Property**

Milliseconds to wait for more output from RunProgram / RunProgramEx.

**Syntax**

`Options.RunProgramOutputWaitTimeout` As Long

**See Also**

Applies to Options object

**7.5.38 RunProgramRedirInputChunkSize Property**

Size in bytes to chunk Run Program / RunProgramEx redirected input text.

**Syntax**

`Options.RunProgramRedirInputChunkSize` As Long

**See Also**

Applies to Options object

**7.5.39 ScriptTypeLibraries Property**

ProgIDs and/or GUIDs of type libraries to load for scripting (separate multiple with newline characters, optionally append;major;minor to specify library version).

**Syntax**

`Options.ScriptTypeLibraries` As String

**See Also**

Applies to Options object

**7.5.40 SetProjectDirectory Property**

Whether to set the current directory for the process to the path containing the project file when a project is loaded or saved.

**Syntax**

`Options.SetProjectDirectory` As Boolean

**See Also**

Applies to Options object

#### 7.5.41 StripLogLinefeedChar Property

Whether to strip newline and non-printable characters when logging to a file.

##### Syntax

```
Options.StripLogLinefeedChar As Boolean
```

##### See Also

Applies to Options object

#### 7.5.42 TextLogSkippedSteps Property

Whether to log skipped steps to text log file.

##### Syntax

```
Options.TextLogSkippedSteps As Boolean
```

##### See Also

Applies to Options object

#### 7.5.43 TextLogStepEvents Property

When to log step events to text log file.

##### Syntax

```
Options.TextLogStepEvents As Long
```

-1 = Never

0 = Only when step logs output

1 = Always

##### See Also

Applies to Options object

#### 7.5.44 TextLoggerCacheMessages Property

Specifies the number of messages to cache before writing to the build log file for text file log format. A value of 0 disables caching of messages by count.

##### Syntax

```
Options.TextLoggerCacheMessages As Long
```

##### See Also

Applies to Options object

### 7.5.45 TextLoggerCacheSeconds Property

Specifies the number of seconds to cache messages before writing to the build log file for text file log format. A value of 0 disables caching of messages by time.

#### Syntax

```
Options.TextLoggerCacheSeconds As Long
```

#### See Also

Applies to Options object

### 7.5.46 UniqueEncryptionIV Property

By default this property is False and encrypted step and macro properties are stored encrypted using the AES algorithm with a *static* initialization vector. For highest security, this property can be set to True to use a *unique* initialization vector every time.

*Note:* One side effect of settings this property to True is that each time a project is saved, the encrypted value of properties in the .bld file will change, which can be problematic for tracking differences in source control systems.

#### Syntax

```
Options.UniqueEncryptionIV As Boolean
```

#### See Also

Applies to Options object

### 7.5.47 UseLogonCreateProcessAsUser Property

By default, RunProgram / RunProgramEx and derived actions calls the CreateProcessWithLogonW API when specifying alternate user credentials for the process. If this option is set to *True*, LogonUser and CreateProcessAsUser will be called instead. This can be useful to work around limitations of CreateProcessWithLogonW.

#### Syntax

```
Options.UseLogonCreateProcessAsUser As Boolean
```

#### See Also

Applies to Options object

### 7.5.48 UseLogonOptionWithSyncBuildEx Property

When SyncBuildEx is used to start a build with a *logonNetCredentialsOnly* parameter value of False, and if UseLogonCreateProcessAsUser is False (the default), the logon flags in the call to CreateProcessWithLogonW API will normally be set to LOGON\_WITH\_PROFILE. If the *UseProfileOptionWithSyncBuildEx* property is set to False, a logon flags value of 0 will be passed instead. This can be useful if the error "A logon request contained an invalid logon type value" is encountered.

#### Syntax



`Options.UseProfileOptionWithSyncBuildEx` As Boolean

### See Also

Applies to Options object

## 7.5.49 UseUTF8ForConsoleApps Property

By default, the console application and RunProgram / RunProgramEx and derived actions use the current code page for processing of program input and output. If this option is set to *True*, UTF-8 conversions will be performed instead. Enabling this option can improve capture and logging of Unicode text from programs that are called from a build, but it requires Windows Vista or later (console program output may not be captured properly for older versions of Windows). When running the console application from a Command Prompt, a Unicode font must also be used for proper display of Unicode text.

### Syntax

`Options.UseUTF8ForConsoleApps` As Boolean

### See Also

Applies to Options object

## 7.5.50 UseUSEnglishLocaleForScriptEngine

Determines whether the default or the US English locale is used for script code. If *False*, the currently configured Windows locale is used for error messages, date/time formatting and conversion, etc. If *True*, the US English locale is used for error messages, date/time formatting and conversion, etc.

### Syntax

`Options.UseUSEnglishLocaleForScriptEngine` As Boolean

### See Also

Applies to Options object

## 7.5.51 WriteBOM Property

Whether to include byte order mark on XML files that are written.

### Syntax

`Options.WriteBOM` As Boolean

### See Also

Applies to Options object

## 7.5.52 XMLLoggerCacheSeconds Property

Specifies the number of seconds to cache messages before writing to the build log file for XML file log format. A value of 0 disables caching of messages by time.

### Syntax

`Options.XMLLoggerCacheSeconds` As Long

**See Also**

Applies to Options object

**7.5.53 XMLLoggerCacheMessages Property**

Specifies the number of messages to cache before writing to the build log file for XML file log format. A value of 0 disables caching of messages by count.

**Syntax**

```
Options.XMLLoggerCacheMessages As Long
```

**See Also**

Applies to Options object

**7.5.54 Load Method**

Loads application options from a file.

**Syntax**

```
Options.Load(ByVal filename As String)
```

**Arguments**

*Options*

**Options** object

*filename*

Optional. The filename to load the options from or the last-loaded options file if not specified.

**See Also**

Applies to Options object

**7.5.55 Save Method**

Saves application options to a file.

**Syntax**

```
Options.Save(ByVal filename As String = "")
```

**Arguments**

*Options*

**Options** object

*filename*

Optional. The filename to save the options to (saves to the last loaded filename if not specified).

**See Also**

Applies to Options object

## 7.6 Project Object

Holds all information for the current Visual Build project. The global Project item is available to all script code

### Properties

Comments | IsModified | Macros | Name | Parent | Password | Scripts | **Steps**

### Methods

FindStep | Load | New | Save

### See Also

Application object

### 7.6.1 Macros Property

Provides the **IMacros** interface of the Macros collection.

#### Syntax

```
Project.Macros As IMacros
```

#### Arguments

```
Project  
    Project object
```

#### See Also

Applies to Project object

### 7.6.2 Name Property

The filename of the current project. Read-only.

#### Syntax

```
Project.Name As String
```

#### See Also

Applies to Project object

### 7.6.3 Parent Property

Returns the parent object of the current object. Read-only.

#### Syntax

```
object.Parent As Object
```

## Arguments

*object*

An object expression that evaluates to an object in the Applies To list.

## See Also

Applies to Options object, Project object, Steps collection, Macros collection, Step object, Macro object, Scripts collection, Script object

### 7.6.4 Password Property

Returns or sets a key used to encrypt password properties in the project file when loading or saving a project.

#### Syntax

```
object.Password As String
```

#### Arguments

*object*

An object expression that evaluates to an object in the Applies To list.

#### See Also

Applies to Project object

### 7.6.5 Scripts Property

Provides the **IScripts** interface of the Scripts collection.

#### Syntax

```
Project.Scripts As IScripts
```

#### Arguments

*Project*

**Project** object

#### See Also

Applies to Project object

### 7.6.6 Steps Property

Provides the **ISteps** interface of the **Steps** collection.

#### Syntax

```
Project.Steps(ByVal type As StepTypeEnum) As ISteps
```

## Arguments

*Project*

**Project** object

*type*

Required. The step collection to retrieve. The values for *type* are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldStepMain</b>	0	The project steps collection.
<b>vbldStepSubroutine</b>	1	The subroutine steps collection.
<b>vbldStepFailure</b>	2	The failure steps collection.
<b>vbldStepGlobalSubroutine</b>	3	The global subroutine steps collection.

## See Also

Applies to Project object

### 7.6.7 Comments Property

Retrieves or updates the comments stored with a project.

#### Syntax

*Project*.Comments As String

## Arguments

*Project*

**Project** object

## See Also

Applies to Project object

### 7.6.8 IsModified Property

Returns the modification status of the project or collection. Read-only.

#### Syntax

*Project*.IsModified As Boolean

## Arguments

*Project*

**Project** object

## See Also

Applies to Project object, Macros collection, Steps collection, Scripts collection

### 7.6.9 FindStep Method

Search for a step by name and return the 0-based index of the first matching step or -1 if not found.

## Syntax

```
Project.FindStep(ByVal sname As String, ByVal stype As StepTypeEnum, ByVal  
maxIndent As Long = -1) As Long
```

## Arguments

*Project*

**Project** object

*sname*

Required. The name of the step to search for. A non-case-sensitive search is performed.

*stype*

Required. The step collection to search.

*maxIndent*

Optional. The maximum step indent to match or -1 to search regardless of indentation.

## See Also

Applies to Project object

### 7.6.10 Load Method

Loads the given project file.

## Syntax

```
Project.Load(ByVal filename As String, ByVal password As String = "")
```

## Arguments

*Project*

**Project** object

*filename*

Required. The filename of the project to load.

*password*

Optional. The password used to encrypt values in the project.

## See Also

Applies to Project object

### 7.6.11 Save Method

Saves the current project file.

## Syntax

```
Project.Save(ByVal filename As String = "", ByVal includeStatus As
```

```
ProjectStatusSaveOptionsEnum = vbldSaveStatusUseDefault, ByVal  
includeExpandState As Boolean = False)
```

## Arguments

*Project*

**Project** object

*filename*

Optional. The filename to save the project to (replaces the existing project file if not provided).

*includeStatus*

Optional. Whether to persist the build status with the project file. The values for *includeStatus* are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldSaveStatusUseDefault</b>	0	Use the setting configured in the Options object.
<b>vbldSaveStatusInclude</b>	1	Save each step's build status in the project file.
<b>vbldSaveStatusExclude</b>	2	Do not save the step build status in the project file.

*includeExpandState*

Optional. Determines whether the expand/collapse state of each step is written to the project file.

## See Also

Applies to Project object

## 7.6.12 New Method

Clears the contents of the current project object.

### Syntax

```
Project.New(ByVal addGroupStep = False)
```

## Arguments

*Project*

**Project** object

*addGroupStep*

Determines whether a single Group step action is added to the Project steps.

## See Also

Applies to Project object

## 7.7 Script Object

Holds information for Visual Build script.

### Properties

Parent | Type | Language | **Value**

### See Also

Scripts collection

### 7.7.1 Language Property

Returns the language of the script code. Read-only.

#### Syntax

```
Script.Language As String
```

#### Arguments

```
Script  
    Script object
```

#### See Also

Applies to Script object

### 7.7.2 Value Property

Sets or returns the script code.

#### Syntax

```
Script.Value As String
```

#### Arguments

```
Script  
    Script object
```

#### See Also

Applies to Script object

## 7.8 Scripts Collection

Holds all scripts of a given type.

### Properties

Parent | Count | IsModified | **Item** | Name | Type | \_NewEnum



## Methods

Add | Remove | Clear

## See Also

Project object

### 7.8.1 Type Property

Returns the type of scripts in the collection. Read-only.

#### Syntax

```
object.Type As ScriptTypeEnum
```

#### Arguments

*object*

An object expression that evaluates to an object in the Applies To list.

The return values are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldScriptAll</b>	-1	A collection of all script of the types below, concatenated together.
<b>vbldScriptTemporary</b>	0	Temporary scripts (not persisted between sessions; created or loaded during a build).
<b>vbldScriptProject</b>	1	Project scripts (saved with the project).
<b>vbldScriptGlobal</b>	2	Global scripts. By default, global scripts are loaded/saved from/to <code>VisBuildPro.scripts</code> in the configuration files path.
<b>vbldScriptSystem</b>	3	System scripts (stored in <code>VisBuildPro.System.scripts</code> ; read-only).

## See Also

Applies to Scripts collection, Script object

### 7.8.2 Item Property

Returns the **Script** object for the given language from the collection.

#### Syntax

```
Scripts.Item(ByVal language As String) As IScript
```

#### Arguments

*Scripts*

**Scripts** collection

*language*

Required. Language of the script to retrieve. Languages are compared without case sensitivity.

*Return Value*

Returns the **Script** object if found or Nothing if not found.

### See Also

Applies to **Scripts** collection

## 7.8.3 Add Method

Adds **Script** object for a new language to the collection.

### Syntax

```
Scripts.Add(ByVal language As String, ByVal code As String = "") As IScript
```

### Arguments

*Scripts*

**Scripts** collection

*language*

Required. Language of the script code. Names are compared without case sensitivity.

*code*

Optional. Script code.

*Return Value*

The new **Script** object. If the script already exists, its code is updated to the specified value.

### See Also

Applies to Scripts collection

## 7.8.4 Remove Method

Removes a **Script** object for a given language from the collection.

*Note:* Any references to the script should be disposed of (`Set objScript = Nothing`) before removing.

### Syntax

```
Scripts.Remove(ByVal language As String) As Boolean
```

### Arguments

*Scripts*

**Scripts** collection

*language*

Required. Language of the script code. Names are compared without case sensitivity.

*code*

Optional. Script code.

*Return Value*

True if the script was found and removed or False if the script was not found.

**See Also**

Applies to Scripts collection

## 7.9 Step Object

Holds information for a Visual Build step.

**Properties****Standard properties**

Parent | Type | Action | BuildFailureStepsOnFailure | BuildStatus | Checked | ContinueOnFailure | Description | Expanded | FailureSubroutine | ID | Indent | Index | **Name** | NoLogging | ExpProperty | Property | Property2 | PropertyEx | Properties | RuleExpression | RuleComparison | RuleCompareTo | RuleRepeat | Script

**Properties specific to Run Program action**

Command | StartIn | OutputFrom | OutputFile | DelOutputFile | Window | Wait | SuccessCodes

**See Also**

Steps collection

### 7.9.1 Action Property

Sets or returns the name of the custom action or user action for the step. For built-in actions, this value is the name of the action as documented in the help.

**Syntax**

*Step*.Action As String

**Arguments**

*Step*  
**Step** object

**See Also**

Applies to Step object

### 7.9.2 BuildStatus Property

Returns the build status of the step.

**Syntax**

*Step*.BuildStatus As StepStatusEnum

**Arguments**

*Step*

**Step** object

The possible return values are:

<b><u>Constant</u></b>	<b><u>Value</u></b>	<b><u>Description</u></b>
<b>vbldStepStatSucceeded</b>	0	The step was successfully built.
<b>vbldStepStatFailed</b>	1	The step failed.
<b>vbldStepStatAborted</b>	2	The build was aborted by the user while processing the step.
<b>vbldStepStatSkipped</b>	3	The step was skipped.
<b>vbldStepStatInProgress</b>	4	The step is currently being built.
<b>vbldStepStatMacroError</b>	5	An error occurred expanding macros or script in a step property.
<b>vbldStepStatPartial</b>	6	An iterating step (repeating build rule or Process Files action) completed an iteration but is not yet complete.
<b>vbldStepStatTerminated</b>	7	The step was terminated.

**See Also**

Applies to Step object

**7.9.3 Checked Property**

Sets or returns the check status of the step (whether to include the step in a build).

**Syntax**

*Step.Checked* As Boolean

**Arguments**

*Step*  
**Step** object

**See Also**

Applies to Step object

**7.9.4 ContinueOnFailure Property**

Sets or returns the continue on failure setting of the step.

**Syntax**

*Step.ContinueOnFailure* As Boolean

**Arguments**

*Step*  
**Step** object

**See Also**

Applies to Step object

### 7.9.5 Description Property

Sets or updates the item's description.

#### Syntax

```
object.Description As String
```

#### Arguments

*object*

An object expression that evaluates to an object in the Applies To list.

#### See Also

Applies to Step object, Macro object

### 7.9.6 Expanded Property

Sets or returns the expand/collapse state of the step. This property only has meaning in the GUI app when the *Save expand/collapse state of steps in project file* user option is turned on.

#### Syntax

```
Step.Expanded As Boolean
```

#### Arguments

*Step*

**Step** object

#### See Also

Applies to Step object

### 7.9.7 ExpProperty Property

Returns a step property with all macros and script expanded.

#### Syntax

```
Step.ExpProperty(ByVal builder As Builder, ByVal name As String, Optional  
ByVal varType As VarType = vbString) As Variant
```

#### Arguments

*Step*

**Step** object

*builder*

**Builder** object (required).

*name*

Required. Name of step property to retrieve (case-sensitive matching). If the property does not exist, Missing is returned.

*Note:* To determine the name of each property for a built-in custom action, create a step for that action in the GUI, save the project, open the .bld file in a text editor, and examine the elements under that step element in the file.

*varType*

Optional (defaults to vbString/VT\_BSTR). Variant type to convert expanded value to before returning. Use vbBoolean/VT\_BOOL (11) for checkbox properties, vbLong/VT\_I4 (3) for radio button or drop-down combo properties, and vbString/VT\_BSTR (8) for all other property types.

### See Also

Applies to Step object

## 7.9.8 FailureSubroutine Property

Sets or returns the name of the failure subroutine to build if the step fails to build and the BuildFailureStepsOnFailure property is set. An empty value causes all failure subroutine steps to build.

### Syntax

```
Step.FailureSubroutine As String
```

### Arguments

*Step*

**Step** object

### See Also

Applies to Step object

## 7.9.9 ID Property

Returns the unique ID of the step (read-only).

### Syntax

```
Step.ID As Long
```

### Arguments

*Step*

**Step** object

### See Also

Applies to Step object

## 7.9.10 Indent Property

Sets or returns the 0-based indent level of the step.

**Syntax**

*Step.Indent* As Long

**Arguments**

*Step*  
**Step** object

**See Also**

Applies to Step object

**7.9.11 Index Property**

Returns the 0-based index of the step (read-only).

**Syntax**

*Step.Index* As Long

**Arguments**

*Step*  
**Step** object

**See Also**

Applies to Step object

**7.9.12 BuildFailureStepsOnFailure Property**

Sets or returns the flag determine if failure steps will be built if the step fails to build.

**Syntax**

*Step.BuildFailureStepsOnFailure* As Boolean

**Arguments**

*Step*  
**Step** object

**See Also**

Applies to Step object

**7.9.13 Name Property**

Sets or returns the name of the step.

**Syntax**

*Step.Name* As String

**Arguments**

*Step*

**Step** object

### See Also

Applies to Step object

## 7.9.14 NoLogging Property

Sets or returns the no logging setting of the step.

### Syntax

```
Step.NoLogging As Boolean
```

### Arguments

*Step*

**Step** object

### See Also

Applies to Step object

## 7.9.15 Property Property

Sets or returns a raw property value by name.

### Syntax

```
Step.Property(ByVal name As String) As Variant
```

Sets or returns the property value. For non-array properties, it can be a Boolean value (check fields), a Long value (drop-down list combo or radio button fields), or a String value (string fields or properties holding a field override value). For array properties (grid fields), the returned value is a variant array with a lower bound of 0.

To determine the name of each property for a built-in custom action, create a step for that action in the GUI, save the project, open the .bld file in a text editor, and examine the elements under that step element in the file. The type of a property indicates the data type of the value (3 = integer, 8 = string, 11 = Boolean).

### Arguments

*Step*

**Step** object

*name*

Required. Name of step property to set or retrieve (case-sensitive matching). If the property does not exist, Missing is returned.

### See Also

Applies to Step object



### 7.9.16 Property2 Property

Sets or returns a raw property value by name, specifying encryption.

#### Syntax

```
Step.Property2(ByVal name As String, Boolean encrypt) As Variant
```

Sets or returns the property value (if True is passed for the *encrypt* parameter when getting, any encrypted values will be returned with the value masked). For non-array properties, it can be a Boolean value (check fields), a Long value (drop-down list combo or radio button fields), or a String value (string fields or properties holding a field override value). For array properties (grid fields), the returned value is a variant array with a lower bound of 0.

To determine the name of each property for a built-in custom action, create a step for that action in the GUI, save the project, open the .bld file in a text editor, and examine the elements under that step element in the file. The type of a property indicates the data type of the value (3 = integer, 8 = string, 11 = Boolean).

#### Arguments

*Step*

**Step** object

*name*

Required. Name of step property to set or retrieve (case-sensitive matching). If the property does not exist, Missing is returned.

#### See Also

Applies to Step object

### 7.9.17 PropertyEx Property

Returns the raw property value, converting to the specified type.

#### Syntax

```
Step.PropertyEx(ByVal name As String, ByVal varType As VarType) As Variant
```

Returns the property value. For non-array properties, it can be a Boolean value (properties for checkbox fields), a Long value (properties for drop-down list combo or radio button fields), or a String value (properties for string fields or properties holding a field override value). For array properties, the returned value is a variant array with a lower bound of 0.

To determine the name of each property for a built-in custom action, create a step for that action in the GUI, save the project, open the .bld file in a text editor, and examine the elements under that step element in the file.

#### Arguments

*Step*

**Step** object

*name*

Required. Name of step property to set or retrieve (case-sensitive matching). If the property does not exist, Missing is returned.

*varType*

Required. Variant type to convert expanded value to before returning. Use vbBoolean for checkbox properties, vbLong for radio button or drop-down combo properties, and vbString for all other property types.

### See Also

Applies to Step object

## 7.9.18 Properties Property

Returns an array containing the names of all properties in the step (read-only).

### Syntax

```
Step.Properties As Variant
```

### Arguments

*Step*  
Step object

The following code demonstrates how to iterate over all step properties, logging their names and values.

```
For Each name in Step.Properties
    Builder.LogMessage name & " = " & Step.Property(name)
Next
```

### See Also

Applies to Step object

## 7.9.19 RuleComparison Property

Sets or returns the type of conditional build rule for the step.

### Syntax

```
Step.RuleComparison As RuleComparisonEnum
```

### Arguments

*Step*  
Step object

The values for the property are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbldRuleNone</b>	-1	Always perform the step (no conditional rule).
<b>vbldRuleUndefined</b>	0	Build the step only if the macro in RuleExpression is undefined.
<b>vbldRuleDefined</b>	1	Build if the macro in RuleExpression is defined.
<b>vbldRuleContains</b>	2	Build if RuleExpression contains the expression in RuleCompareTo property

<b>vbldRuleEqual</b>	3	Build if RuleExpression does not contain the expression in RuleCompareTo
<b>vbldRuleNotEqual4</b>		Build if RuleExpression equals RuleCompareTo
<b>vbldRuleDoesNot Contain</b>	5	Build if RuleExpression does not equal RuleCompareTo
<b>vbldRuleTrue</b>	6	Build if RuleExpression is true (a non-zero number or the string <i>True</i> ).
<b>vbldRuleFalse</b>	7	Build if RuleExpression is true (0 or the string <i>False</i> ).

### See Also

Applies to Step object

## 7.9.20 RuleExpression Property

Sets or returns the rule expression for a conditional build rule.

### Syntax

```
Step.RuleExpression As String
```

### Arguments

*Step*  
**Step** object

### See Also

Applies to Step object

## 7.9.21 RuleCompareTo Property

Sets or returns the comparison value for a conditional build rule.

### Syntax

```
Step.RuleCompareTo As String
```

### Arguments

*Step*  
**Step** object

### See Also

Applies to Step object

## 7.9.22 RuleRepeat Property

Sets or clears whether the rule will repeat while the condition is true.

### Syntax

```
Step.RuleRepeat As Boolean
```

## Arguments

*Step*  
**Step** object

## See Also

Applies to Step object

### 7.9.23 Script Property

Sets or returns the script code associated with the step.

## Syntax

```
Step.Script As String
```

## Arguments

*Step*  
**Step** object

## See Also

Applies to Step object

### 7.10 Steps Collection

Holds all steps of a given type.

## Properties

Parent | Type | IsModified | **Item** | Count | \_NewEnum

## Methods

Add | Clear | Find | Load | Remove | Save

## See Also

Steps property

#### 7.10.1 Type Property

Returns the type of steps in the collection. Read-only.

## Syntax

```
object.Type As StepTypeEnum
```

## Arguments

*object*  
An object expression that evaluates to an object in the Applies To list.

The return values are:

<u>Constant</u>	<u>Value</u>	<u>Description</u>
<b>vbIdStepMain</b>	0	The project steps collection.
<b>vbIdStepSubroutine</b>	1	The subroutine steps collection.
<b>vbIdStepFailure</b>	2	The failure steps collection.
<b>vbIdStepGlobalSubroutine</b>	3	The global subroutine steps collection.

### See Also

Applies to Steps collection, Steps property, Step object

## 7.10.2 Item Property

Returns the given **Step** object from the collection.

### Syntax

```
Steps.Item(ByVal index As Long) As IStep
```

### Arguments

*Steps*

**Steps** collection

*index*

Required. 0-based index of the step to return.

*Return Value*

Returns the Step object if found or raises an error if the index is not valid.

### See Also

Applies to Steps collection

## 7.10.3 Count Property

Returns the number of items in the collection. Read-only.

### Syntax

```
object.Count As Long
```

### Arguments

*object*

An object expression that evaluates to an object in the Applies To list.

### See Also

Applies to Steps collection, Macros collection, Scripts collection,

### 7.10.4 **\_NewEnum Property**

Returns an enumerator object for items in the collection.

#### **Syntax**

```
object._NewEnum As IEnumVARIANT
```

#### **Arguments**

*object*

An object expression that evaluates to an object in the Applies To list.

#### **See Also**

Applies to Steps collection, Macros collection, Scripts collection,

### 7.10.5 **Add Method**

Adds a **Step** object to the collection.

#### **Syntax**

```
Steps.Add(ByVal action As String, ByVal index As Long = -1) As IStep
```

#### **Arguments**

*Steps*

**Steps** collection

*action*

Required. Name of the custom action or user action for the step. For built-in actions, this value is the name of the action as documented in the help.

*index*

Optional. 0-based index to insert the step at. If not provided, inserts at the end of the collection.

#### **See Also**

Applies to Steps collection

### 7.10.6 **Clear Method**

Removes all items from the collection.

#### **Syntax**

```
object.Clear
```

#### **Arguments**

*object*

An object expression that evaluates to an object in the Applies To list.

#### **See Also**

Applies to Steps collection, Macros collection, Scripts collection,

### 7.10.7 Find Method

Search for a step by its ID and return the matching step or Nothing if not found.

#### Syntax

```
Steps.Find(ByVal id As Long) As Step
```

#### Arguments

*Steps*

**Steps** collection

*id*

Required. The ID of the step to search for.

#### See Also

Applies to Steps collection

### 7.10.8 Remove Method

Removes a **Step** object from the collection.

*Note:* Any references to the step should be disposed of (`Set objStep = Nothing`) before removing.

#### Syntax

```
Steps.Remove(ByVal index As Long)
```

#### Arguments

*Steps*

**Steps** collection

*index*

Required. 0-based index of the step to remove.

#### See Also

Applies to Steps collection

## 7.11 WScript Object

A helper object to provide compatibility with [WSH](#) script code. It's actually an alias for the Builder object, so all methods on each object are available on both.

#### Methods

CreateObject | Echo | Sleep

**See Also**

Builder object

**7.11.1 CreateObject Method**

Creates a COM object.

**Syntax**

```
object.CreateObject(ByVal progID As String) As Object
```

**Arguments**

*progID*

String value indicating the programmatic identifier (ProgId) of the object to be created.

**See Also**

Applies to WScript object

**7.11.2 Echo Method**

Logs the specified message (equivalent to Builder.LogMessage).

**Syntax**

```
object.Echo(ByVal message As String)
```

**Arguments**

*message*

String to log

**See Also**

Applies to WScript object

**7.11.3 Sleep Method**

Suspends the build thread for the specified amount of time.

**Syntax**

```
object.Sleep(ByVal time As Long)
```

**Arguments**

*time*

Amount of time (in milliseconds) to pause.

**See Also**

Applies to WScript object



## 7.12 Threading

The Builder component is marked as **Free threaded**, meaning that it and the build thread that it creates when building will operate in the multi-threaded apartment (MTA) of the calling process. An alternate Builder component marked **Both threaded**, which can exist in a single-thread apartment (STA) or the MTA, is also available by using a ProgID of **VisBuildSvr9.BuilderSTA** or **VisBuildSvr.BuilderSTA**. Calling the Both version from an STA thread can yield better performance, but any event handling in the calling code of a GUI application **must** marshal all events back to the GUI thread, and any custom actions called from the build must support running from an STA.

Prior to v6.3, the GUI app and Console app always created the Free threaded builder component, so all built-in and custom actions marked as Both threaded that were created during a build would get created in the MTA. For performance reasons, starting with v6.3, the GUI and Console app now both enter an STA and use the Both threaded component by default, so actions marked as Both threaded will now get created in an STA. This can be overridden via the `/mta` command-line flag to force the older MTA threading mode. This might be desired if a custom action component is marked as Both threaded but actually needs to be created in the MTA (for instance, if it calls a non-blocking Wait API). The Watches, Call Stack, and Macros panes are also updated immediately during a build with MTA mode, but in STA mode these panes are not updated during a build unless the build is paused or debugged (singled-stepped) or the related option is enabled.

See this article for more details on COM threading models.

## 8 Purchase & Support

### 8.1 Order Info

All orders are subject to the License Agreement. The registered version never expires, and licensed users can receive technical support and products updates.

Email delivery of Visual Build licenses is available for credit card orders placed via the Internet from the [Kinook Software](#) web site. If you prefer, Visual Build can also be ordered by telephone, mail, or by corporate purchase order. Please visit the [web site](#) for pricing and ordering details.

Customer information is considered confidential and will not be shared or distributed to any third party.

### 8.2 How To Register

Visual Build can be registered using the Registration dialog, which is accessed by starting Visual Build and selecting the *Register* item on the *Help* menu.

If you paid the Visual Build licensing fee and received product registration information from Kinook Software, Inc. or an authorized reseller, copy and paste or drag and drop the registration information as specified in the instructions you received. If entered correctly, the registration info will be updated in the About box and any evaluation limitations will be removed. Please save this information for possible use with future upgrades or in case you need to re-install Visual Build.

If you downloaded an evaluation version of Visual Build, or if you received this version of Visual Build on a disk or CD with a book or with other hardware or software, and you have not paid the Visual Build licensing fee, you are licensed to use Visual Build for evaluation purposes only for a period of 30 days.

For information on purchasing a license to use Visual Build for non-evaluation purposes, see the Order Information section.

## 8.3 Technical Support

To verify that you have the most recent version of Visual Build, select Help | Check for Updates on the menu or visit the Kinook Software [download page](#).

When reporting any problems encountered, please ZIP and send **all** of the following information to support@kinook.com:

- 1) The info from Help | About | Install Info on the menu.
- 2) If the problem involves interaction with another program (i.e., Visual Studio, SourceSafe, etc.), the version and service pack of that application.
- 3) A clear description of the problem encountered and screen shots of any error message windows. To create a screen shot, press Alt+PrtScn with the error window focused, then Win+R (to show the Run dialog), mspaint and Enter (to start Paint), Ctrl+V to paste the image, and then save the file (preferably in PNG format).
- 4) Detailed steps to reproduce the problem.
- 5) A .bld file (ideally that can be built by us) and any other files needed for #4.
- 6) A build log file.

If the ZIP file to send is larger than 1MB, send it via a free service such as <http://www.mailbigfile.com> or <http://www.sendthisfile.com>.

We will need all of the above information in order to investigate the problem. All information provided will be kept strictly confidential to Kinook technical support. Thank you.

*Note:* You can press F1 at any time while Visual Build is active for context sensitive help.

You can also visit the [Visual Build Forum](#) for answers to common questions and to post questions to the Visual Build user community.

## 8.4 Revision History

Visit the [Kinook Software web site](#) for details on product history.

# Index

## - % -

% 48, 343, 344  
%% 48, 70, 344

## - & -

& 64

## - \* -

\* 299

## - . -

.action 219, 301  
.aip 129  
.bat 193  
.bdsgroup 82  
.bdsproj 82  
.bpgr 85  
.bpr 82  
.cbproj 82  
.cmd 193  
.com 64  
.cpp 158, 178  
.cs 158, 178  
.csdproj 158  
.csproj 158, 178  
.deploy 130  
.dll 219  
.dox 116  
.dpk 82  
.dpr 82  
.dproj 82  
.dsp 155  
.dsw 155  
.dtproj 158  
.dwproj 158  
.dxp 118  
.ebg 151  
.ebp 151

.exe 64  
.flprj 120  
.gui 119  
.hgu 122  
.hm2 121  
.hm3 121  
.hmx 121  
.hnd 123  
.hsc 125  
.hsp 126  
.hts 127  
.i2g 136  
.iap 133  
.iap\_xml 133  
.iax 134  
.icproj 158  
.im7 130  
.ini 114  
.ipr 136  
.ise 136  
.ism 136  
.iss 131  
.jpgr 85  
.jpr 85  
.jpx 85  
.jsl 158, 178  
.mpr 134  
.msi 143  
.mxd 140  
.NET 150, 151, 180, 181, 234, 236, 240, 241, 243, 245, 247, 322, 324  
.NET actions 301  
.NET Compact Framework 158, 178  
.NET Framework 180, 181  
.NET Framework SDK Tools 180  
.nsi 141  
.ocx 219  
.olb 219  
.oxygene 158  
.pfb 136  
.proj 82  
.ps1 196  
.rc 155, 158, 178  
.rptproj 158  
.sb5 142  
.sb6 142  
.sb7 142  
.sf7 143

.sh4 123  
 .sh5 123  
 .sh6 123  
 .sh7 123  
 .sln 158  
 .sufproj 143  
 .tin 130  
 .tip 130  
 .tlb 219  
 .user 34  
 .vb 158, 178  
 .vbdproj 158  
 .vbg 151  
 .vbp 151  
 .vbproj 158, 178  
 .vcp 155  
 .vcproj 158, 178  
 .vcw 155  
 .vcxproj 178  
 .vdproj 158  
 .vjsproj 158, 178  
 .wdproj 158  
 .wixproj 158  
 .wse 145  
 .wsi 145  
 .wsm 145  
 .wxs 146  
 .xml 111  
 .xpj 128

**- / -**

/regserver 219  
 /unregserver 219

**- [ -**

[ 48, 304, 343, 344  
 [[ 48, 344

**- \ -**

\\?\ 344

**- ] -**

] 48, 304, 343, 344

]] 48, 344

**- \_ -**

\_BUILD\_PROFILE 24  
 \_NewEnum Property (Steps) 418

**- | -**

| 64

**- < -**

<object> 308, 379

**- > -**

> 64  
 >> 64

**- 3 -**

32-bit 333

**- 6 -**

64-bit 333

**- 7 -**

7z files 73  
 7-Zip 73

**- A -**

abort 34, 41, 63  
 aborted 34  
 about 38  
 accept user input 324  
 Access 217, 320  
 accidental 338  
 AccuRev 248  
 AccuRev Action 248  
 Acresso 133, 136  
 action 219

- action component 40
  - action GUI 39
  - action properties 38
  - Action Property (Step) 407
  - action script 40
  - actions 25, 38, 62, 63, 64, 69, 70, 71, 86, 89, 91, 96, 99, 100, 101, 102, 104, 105, 107, 108, 111, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 125, 126, 127, 128, 129, 130, 131, 133, 134, 136, 140, 141, 142, 143, 145, 146, 151, 155, 158, 166, 178, 180, 181, 182, 183, 184, 185, 188, 189, 190, 192, 195, 198, 199, 200, 201, 206, 210, 212, 214, 217, 219, 220, 221, 222, 223, 224, 225, 226, 228, 230, 248, 250, 252, 260, 262, 264, 265, 266, 270, 272, 273, 276, 278, 280, 284, 285, 287, 290, 291, 293, 294, 295, 297, 299, 300, 301, 303
  - Actions pane 25
  - activate 59
  - Active Directory 217
  - Active Script 304
  - Active Scripting 51
  - ad hoc 34
  - add 45
  - Add Method (Macros) 382
  - Add Method (Scripts) 406
  - Add Method (Steps) 418
  - add step 25
  - AddAsEnvVar Property (Macro) 380
  - AddEx Method (Macros) 383
  - addin 179
  - adding steps 25
  - ADO Action 217
  - ADO\_RS 217
  - Adobe 128
  - ADODB 217
  - advanced 66, 317
  - Advanced Installer 129
  - Advanced Installer Action 129
  - Advanced.bld 317
  - advantages 1
  - AES 45
  - affinity 66
  - age 94
  - aip 129
  - Al.exe 180
  - Alienbrain 250
  - Alienbrain Action 250
  - Altiris 145
  - AlwaysShowGUIApps Property (Options) 385
  - Amazon 218
  - Amazon Web Services 218
  - Ant 233
  - Apache Ant 233
  - Apache Maven 239
  - Apache Subversion 280
  - App Packager 180
  - App Property (Builder) 356
  - appearance 47
  - append 45
  - application 33
  - application data 334
  - Application Object 347, 348, 349, 350, 351, 352, 353, 354
  - application options 41, 42, 43, 44
  - applications 220
  - archive 107
  - ARJ 73
  - array 64
  - ask question 318
  - ASP .NET 158
  - ASP.NET 324
  - Aspnet\_regiis.exe 180
  - assemblies 181
  - AT service 330
  - ATTRIB 20
  - attributes 107
  - Authenticode 108
  - auto complete 61
  - auto-completion 51
  - automate 325
  - automated builds 325, 326, 328, 329
  - automation 319, 346
  - Avatar Software 140
  - Avid Technology 250
  - AWS 218
  - Aximp.exe 180
  - Azure 219
- B -**
- back 337
  - backup 57, 318, 324
  - bat 193
  - batch files 193, 317, 340
  - Bazaar 252
  - Bazaar Action 252
  - BCB 20
  - BCBDIR 20

- BDS 82
  - bds.exe 84
  - bdsgroup 82
  - bdsproj 82
  - BJB 20
  - BJBDIR 20
  - bld 335
  - bldx 335
  - Blu-Ray 89
  - BOM 44
  - boolean 409
  - Borland 82, 85
  - Borland Developer Studio 82
  - Borland StarTeam 86
  - bpgr 85
  - bpr 82
  - bracket chars 344
  - brackets 344
  - breakpoint 27, 36
  - breakpoints 36
  - browse 48, 55
  - build 1, 23, 34, 35, 63, 181, 182, 190
  - build condition 53
  - build details 36
  - build errors 337
  - build events 305
  - build log 37
  - build options 59
  - build output 30
  - build overview 34
  - build position 34
  - build process 34
  - build profile 24, 35
  - build rules 53
  - build status 34, 323
  - build view 49
  - build.exe 66
  - BuildBot 332
  - BuildConsole.exe 162
  - BuildDone Event 305
  - BuildDone Event (Builder) 377
  - Builder Object 354, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 371, 372, 374, 375, 376, 377, 378
  - BuildFailureStepsOnCancel Property (Options) 386
  - BuildFailureStepsOnFailure Property (Step) 411
  - BuildIndex Property (Builder) 356
  - building 34, 36
  - BuildLauncher 324
  - BuildMessage Event (Builder) 378
  - build-only 45
  - BuildStarting Event 305
  - BuildStarting Event (Builder) 376
  - BuildStatus Property (Step) 407
  - BuildThread Property (Builder) 356
  - burn CD 89, 318
  - Burn CD/DVD Action 89
  - burn DVD 89, 318
  - BURN\_FILE\_COUNT 89
  - BURN\_TOTAL\_SIZE 89
  - buttons 45
  - byte order mark 44
  - BZIP2 files 73
- C -**
- C# 158, 180, 324
  - C# actions 301
  - C#Builder 82
  - C++Builder 82, 318
  - CAB 73
  - CacheActions Property (Options) 386
  - call project 71
  - call stack 32
  - call subroutine 70
  - CallStack2 Property (Builder) 357
  - CallStepStartingScriptEventOnSkippedStepEvents Property (Options) 386
  - cancel 41, 63
  - CancelEvent Property (Builder) 357
  - candle.exe 146
  - Caphyon 129
  - CaseSensitiveBuildRuleComparisons Property (Options) 386
  - Caspol.exe 180
  - Category Property (Macro) 379
  - cbproj 82
  - CD 89, 318
  - CD burning 318
  - CD-R 89
  - CD-RW 89
  - Cert2spc.exe 180
  - certificates 108
  - Certmgr.exe 180
  - chain 71, 317
  - Chain.bld 317

- change 338
- change edit password 45
- change password 45
- check for updates 57
- Checked Property (Step) 408
- checksum 97
- Chktrust.exe 180
- CHM 73
- CI 332
- Clear Method (Steps) 418
- ClearCase 253, 320
- ClearCase Action 253
- ClearCase LT 253
- ClearCase.bld 320
- CLEARCASE\_CONFIGSPEC 253
- clearfsimport 256
- cleartool.exe 253
- ClickOnce 159
- clipboard 335
- cmd 193
- code analysis 234
- code completion 51
- code coverage 241
- code signing 108
- CodeGear 82, 318
- CodeGear.bld 318
- colors 60
- columns 59
- COM 64, 219, 346
- COM Register Action 219
- COM server 219
- COM+ 184, 220, 221, 222, 320
- COM+ Application Action 220
- COM+ Component Action 222
- command 64, 254
- command processor 193
- command scripts 193, 340
- command-line 326, 328, 336
- commands 45, 46
- comments 50
- Comments Property (Project) 401
- Comments: Specifies comments to be applied to the command (optional). 249
- Compact Framework 158, 178
- compatibility 16, 18, 20, 21, 22
- completed 34
- completion status 34
- CompletionStatus Property (Builder) 358
- Component Services 220, 222
- ComponentOne 117
- components 222
- compress files 73, 76, 79
- compression 335
- COMPUTERNAME 340
- COMSPEC 340
- conditional build rule 53, 317
- conditional execution 53
- ConfigFilesPath Property (Options) 386
- configpath 326, 328
- configuration file 44
- configuration files 334
- configure 45, 46
- console app 326
- context menus 46
- Context Property (Application) 348
- ContinueOnFailure Property (Step) 408
- continuous builds 320
- continuous integration 320, 332
- ContinuousIntegration.bld 320
- Control key 46, 337
- convert 18
- ConvertActions.bld 18
- ConvertOutputDoubleQuotes Property (Options) 387
- copy 335
- copy expanded 335
- copy files 92
- Copy Files Action 91
- COPYFILES\_COPY\_COUNT 91
- COPYFILES\_DELETE\_COUNT 91
- COPYFILES\_SKIP\_COUNT 91
- Cordbg.exe 180
- Count Property (Steps) 417
- CPIO 73
- create CD 318
- create directory 96
- create folder 96
- create ISO image 89
- create share 308
- create temporary macro 69
- CreateObject Method (WScript) 420
- CreateProcessAsUser 396
- CreateProcessWithLogonW 372, 375, 396
- credentials 372, 375
- cruise control 332
- CruiseControl.NET 332

- CS\_NET 20
  - Csc.exe 180
  - CSharpClient 324
  - csproj 158
  - csUnit 236
  - current build position 34
  - cursor 34
  - Custom Action Details 299
  - custom actions 25, 62, 63, 64, 69, 70, 71, 73, 74, 76, 78, 79, 82, 85, 86, 89, 91, 96, 97, 98, 99, 100, 101, 102, 104, 105, 107, 108, 111, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 125, 126, 127, 128, 129, 130, 131, 133, 134, 136, 140, 141, 142, 143, 145, 146, 151, 155, 158, 166, 178, 180, 181, 182, 183, 184, 185, 188, 189, 190, 192, 193, 195, 196, 198, 199, 200, 201, 206, 210, 212, 214, 217, 219, 220, 221, 222, 223, 224, 225, 226, 228, 230, 248, 250, 252, 260, 262, 264, 265, 266, 270, 272, 273, 276, 278, 280, 284, 285, 287, 290, 291, 293, 294, 295, 297, 299, 300, 301, 303
  - custom logger 325
  - customize 33, 45, 46, 47
  - customizing 303
  - cut & paste 335
  - CVS 260
  - CVS Action 260, 321
  - CVS.bld 321
- D -**
- dark.exe 146
  - database 217, 226, 320
  - DATE 94, 306, 340
  - DATETIME 340
  - dcc32.exe 84
  - DDK 66
  - DEB 73
  - debugging 32, 34, 36, 37, 42, 304, 337
  - debugging actions 301
  - Debugging Property (Builder) 358
  - default properties 301
  - default property 43
  - default script language 41
  - default value 344
  - DefaultScriptEngine Property (Options) 387
  - definition of terms 34
  - delete 45
  - delete files 92, 96
  - Delete Files Action 96
  - delete folder 96
  - delete macro 70
  - delete share 308
  - DELETEYFILES\_COUNT 96
  - DelLogFileOnBuild Property (Options) 387
  - Delphi 20, 82, 318
  - Delphi Prism 158
  - DELPHIDIR 20
  - DelTempMacrosAfterBuild Property (Options) 387
  - deploy 1, 159
  - DeployMaster 130
  - DeployMaster Action 130
  - deployment tool 230
  - derived action 40
  - Description Property (Step) 409
  - desktop shortcut 24, 336
  - devenv 18, 158
  - devenv.com 162
  - DEVENV\_NET 20
  - DevHost 120
  - DevStudio integration 151, 180
  - DEVSTUDIO\_NET\_DIR 180
  - DEVSTUDIODIR 18
  - dictionary 304, 319
  - digital certificates 108
  - digital signatures 108
  - Dimensions 262
  - Dimensions Action 262
  - dir 98
  - directories 307
  - disable logging 318
  - Disco.exe 180
  - disconnect mapped drive 210
  - display options 59, 60
  - dll 219
  - dock 33
  - Doc-O-Matic 116
  - Doc-O-Matic Action 116
  - Doc-To-Help 117
  - Doc-To-Help Action 117
  - Document! X 118
  - Document! X Action 118
  - DOM 304
  - DOSCMD 340
  - DOTNET\_DIR 21, 180
  - DOTNETSDK\_DIR 21, 180
  - double quotes 387
  - doubled characters 344
  - dox 116



dpk 82  
 dpr 82  
 dproj 82  
 Dr.Explain 119  
 Dr.Explain Action 119  
 drag 336  
 drag & drop 336  
 drop 336  
 dsp 155  
 dsw 155  
 dtproj 158  
 DTS 217  
 dual sign 109  
 DVD 89, 318  
 DVD burning 318  
 DVD+R 89  
 DVD-R 89  
 dwproj 158  
 dxp 118  
 dynamic projects 317, 319  
 Dynamsoft 273, 276

## - E -

ebg 151  
 ebp 151  
 echo 43, 388  
 Echo Method (WScript) 420  
 EchoChainedConsoleOutput Property (Options) 388  
 edit password 47  
 editing 33  
 editor 30  
 email 212  
 Embarcadero 82, 158, 318  
 Embedded Visual C++ 155  
 Embedded Visual Tools 155  
 encryption 45, 49, 335, 396  
 EncryptPasswordProperties Property (Options) 388  
 environment 66  
 environment variables 36, 48, 70, 317, 380  
 EnvVarsInSystemMacros Property (Options) 388  
 errors 338  
 escape 43, 308, 344, 388  
 EscapeSpecialCharactersInOutput Property (Options) 388  
 ESX Server 294, 295  
 ESXi 295

evaluate script 344  
 EvaluateRule Method (Builder) 361  
 EVC 155  
 Event Log 42, 44, 329  
 event logging 338  
 event script 52  
 Event Viewer 42, 329, 338  
 events 305  
 Excel 319  
 Exchange 217  
 exclude 339  
 exe 64  
 exists 307  
 exit 63  
 exit code 65, 71, 369  
 exit codes 326, 328  
 expand 59  
 expand macro 344  
 expand macros 46, 335, 338  
 expand property 308, 409  
 Expanded Property (Step) 409  
 expanding menu 47  
 ExpandMacros Function (Application) 350  
 ExpandMacros Method (Builder) 362  
 ExpandMacrosAndScript Function (Application) 351  
 ExpertInstall 130, 131  
 ExpertInstall Action 130  
 explore 30, 48, 55, 57, 337  
 Explorer Interface 336  
 Export .NET type library 181  
 ExpProperty Property (Step) 409  
 expressions 304  
 extended-length path 344  
 extensibility 301  
 extension 179, 335  
 extensions 300  
 extract files 74, 78  
 Extreme Programming 233, 236, 239, 240, 241, 243, 245, 247, 322

## - F -

fail 41  
 FailBuildWhenDoneIfStepsFailed Property (Options) 389  
 failed 34  
 failed step options 54

FailedStep Property (Builder) 358  
 FAILSTEP\_NAME 340  
 FAILSTEP\_OUTPUT 340  
 FAILSTEP\_STATUS 340  
 failure 54  
 failure steps 28  
 FailureSubroutine Property (Step) 410  
 Fast-Help 120  
 Fast-Help Action 120  
 field override 299  
 file checksum 97  
 file contents 344  
 file copy 91, 201  
 file delete 96  
 file exists 307  
 file format 335  
 file globbing 339  
 file locations 44, 334  
 file logging 37, 42, 50, 318  
 file matching 339  
 file sync 91, 201  
 file transfer 201  
 file version 307  
 file viewer 57  
 FILE\_CHECKSUM\_VALUE 97  
 files 73, 74, 76, 78, 79, 89, 91, 96, 100, 101, 102, 104, 105, 107, 111, 114, 115, 200, 307  
 Files.bld 318  
 FileSystemObject 307  
 filter 25  
 find 47  
 find app 308  
 Find Method (Steps) 419  
 find window 308  
 FindMacro Function (Application) 351  
 FindStep Method (Project) 401  
 fix 338  
 Flare 120  
 Flare Action 120  
 Flexera 136  
 FLEXnet 136  
 FLEXnet InstallShield 136  
 flprj 120  
 folder 96  
 folder exists 307  
 folders 307  
 font 60  
 format date 306, 340

format XML 112  
 Fortran 158  
 Fortress 287  
 Fortress Action 287  
 forward 337  
 frames 33  
 front-end 324  
 FTP 201, 318  
 FTP Action 201, 324  
 FTP\_DELETE\_COUNT 201  
 FTP\_HOST\_FINGERPRINT 201  
 FTP\_TRANSFER\_COUNT 201  
 FTPS 201, 204  
 full menu 47  
 full row select 58, 61  
 functions 28  
 Fuslogvw.exe 180  
 FxCop 234

## - G -

GAC 181  
 GAC Install Action 181  
 Gacutil 181  
 Gallio 236  
 General (More) tab 52  
 General tab 52  
 Generate Resource Files 182  
 get file 201, 206  
 get version 178  
 GetFileContents Method (Builder) 363  
 GetFileVersion 307  
 GetProjVer.bld 322  
 Git 264  
 Git Action 264  
 global application data 334  
 Global Assembly Cache 181  
 global macros 342  
 global settings 334  
 global steps 28  
 global subroutines 28  
 globalization 149  
 GlobalSubroutineStepsLoaded Event (Application) 351  
 globbing 339  
 go 337  
 go to step 48  
 goto 63

Gradle 238  
grid settings 58, 61  
gridlines 58, 61  
group 63  
GSX Server 294  
gui 39, 119  
GUI app 328  
GUID 161  
GZ files 73, 78, 79

## - H -

h.exe 170  
hash 97  
Hatteras 170  
HD-DVD 89  
heat.exe 146  
help 422  
Help & Manual Action 121  
Helpinator Action 122  
HelpMaker 123  
HelpMaker Action 123  
HelpNDoc Action 123  
HelpScribble 125  
HelpScribble Action 125  
HelpStudio 126  
HelpStudio Action 126  
hgu 122  
hidden 107  
hide 25, 33, 326  
HideConsole 326  
HidePasswordPropertyValues Property (Options) 389  
history 422  
hm2 121  
hm3 121  
hmx 121  
hsc 125  
hsp 126  
HTML logging 111  
HTML reports 111  
hts 127  
HTTP 206, 318  
HTTP Action 206  
HTTPS 206, 209  
Hudson 332  
HyperText Studio 127  
HyperText Studio Action 127  
Hyper-V 290

## - I -

iap 133  
iax 134  
IBM 253, 285  
icproj 158  
ID Property (Step) 410  
identity 372, 375  
if/else 51  
ignore failure 52  
IIS 223, 224, 230, 320  
IIS Action 224  
IIS Virtual Dir Action 223  
Ilasm.exe 180  
Ildasm.exe 180  
Import .NET type library 183  
include 339  
include in build 27, 34, 41  
Incredibuild 155, 158, 162  
increment version 155  
indent 112  
Indent Property (Step) 410  
Index Property (Step) 411  
Indexing Service 217  
Indigo Byte Systems 119  
Indigo Rose 143  
INI files 101, 114, 340  
Initialize Method (Application) 352  
Initialize Method (Builder) 363  
Inno Setup Action 131  
Inno Setup Preprocessor 131  
Innovasys 118, 126  
input 318  
insert macro 48, 55  
insert step 25  
InstallAnywhere 133  
InstallAnywhere Action 133  
InstallAnywhere.NET 134  
InstallAnywhere.NET Action 134  
installation 14  
InstallAware 134  
InstallAware Action 134  
Installer2Go Action 136  
InstallMaster 145  
InstallMate 130  
InstallShield 136

InstallShield Action 136  
 InstallShield Developer 136  
 InstallShield DevStudio 136  
 InstallShield Express 136  
 InstallShield Professional 136  
 InstallShield StandAlone compiler 138, 144  
 InstallShield X 136  
 Installutil.exe 180  
 integer 409  
 Intel 158  
 Intel C++ Compiler 158  
 Intel Fortran 158  
 Intellisense 51  
 intercept vector 396  
 internet 318  
 Internet Information Services 223, 224  
 Internet Publishing 217  
 introduction 1  
 ipr 136  
 iscc 131  
 ise 136  
 ism 136  
 IsModified Property (Project) 401  
 ISO 73, 89, 318  
 ispp 131  
 ISQL 226  
 iss 131  
 Item Property (Macros) 381  
 Item Property (Scripts) 405  
 Item Property (Steps) 417  
 iterate 34, 317  
 iterating steps 34  
 iterations 34  
 iterative steps 34

## - J -

J# 158  
 Java 233, 238, 239, 319  
 Jazz 285  
 JBuilder 85  
 Jenkins 332  
 JGSoft 125  
 jpg 85  
 jpr 85  
 jpx 85  
 JScript 41, 51, 304, 306

## - K -

key 45, 49  
 keyboard 46  
 keyboard map 46  
 kill process 195  
 KILLPROCESS\_COUNT 195

## - L -

Language Property (Script) 404  
 large icons 47  
 LastStep Property (Builder) 359  
 LASTSTEP\_NAME 340  
 LASTSTEP\_OUTPUT 340  
 LASTSTEP\_STATUS 340  
 launch 30, 48, 55, 57, 337  
 LaunchType Property (Builder) 359  
 layout 33  
 Lc.exe 180  
 license 13, 421  
 light.exe 146  
 LightSwitch 158  
 LinderSoft 142  
 list 64, 317  
 list files 98  
 LISTFILES\_COUNT 98  
 LISTFILES\_SIZE 98  
 literal 344  
 Load Method (Macros) 384  
 Load Method (Options) 398  
 Load Method (Project) 402  
 Load Method (Scripts) 384  
 Load Method (Steps) 384  
 locale 397  
 localization 149, 150  
 locating errors 337  
 locations 44  
 log command-line 66  
 log file caching 395, 396, 397, 398  
 log format 42  
 log level 364  
 log message 63  
 Log Message Action 63, 317  
 log messages 364  
 log off 200

- LogAllFailureStepOutput Property (Options) 389
- LogDefStepProperty Property (Options) 389, 390
- LOGFILE 37, 326, 328, 340
- LogFileContents Method (Builder) 363
- LogFilename Property (Options) 390
- LogFormat Property (Options) 390
- logging 37, 42, 43, 50, 54, 318, 325, 364
- Logging.bld 318
- loglevel 326
- LogLevel Property (Options) 390
- LogMessage Method (Builder) 364
- LogMessage2 Method (Builder) 364
- LogMessageEx Method (Builder) 365
- logon 372, 375
- LogonUser 396
- long 409
- long filenames 344
- loop 64
- LOOP\_COUNT 64
- LOOP\_INDEX 64
- LOOP\_VALUE 64
- looping 53, 308
- LZH 73
  
- M -**
  
- macro 48
- macro count 50
- Macro Object 379, 380
- macro properties 48
- MacroExpandBuildTimeout Property (Options) 391
- MacroExpandTooltipTimeout Property (Options) 391
- MacroModified Event (Application) 354
- macros 29, 48, 53, 338, 340, 342
- Macros Collection 381, 382, 383, 384
- macros file 44
- Macros pane 29
- Macros Property (Application) 349
- Macros Property (Project) 399
- Macrovision 136
- MadCap Software 120
- mage.exe 185
- main screen 23
- main steps 27
- main toolbar 24
- Make Delphi Action 82
- Make JBuilder Action 85
  
- Make VB6 Action 151
- Make VC6 Action 155
- Make VS.NET Action 158
- MakeAppx 180
- makecert 110
- Makecert.exe 180
- makensis 141
- manifest 98
- Manifest Generator 185
- map drive 210
- map network path 210
- MAPDRIVE\_LETTER 210
- mask 99
- masks 339
- match files 99, 339
- matching 339
- Maven 239
- MAX\_PATH 344
- MaxDefaultPropertyLogLength Property (Options) 391
- MaxMacroLengthExpand Property (Options) 391
- MaxStepOutputLength Property (Options) 392
- MaxStopWait Property (Builder) 360
- MbUnit 236
- MD 96
- menu 24
- menu animation 47
- menu bar 24
- menu delay 47
- menus 46
- Mercurial 265
- Mercurial Action 265
- methods of building 34
- Mgmtclassgen.exe 180
- Microsoft 163, 168, 170, 176
- Microsoft Excel 319
- Microsoft Hyper-V 290
- Microsoft Office 319
- Microsoft Virtual PC 291
- Microsoft Virtual Server 293
- Microsoft Visual Studio 151, 158, 179, 180
- MimarSinan 134
- minimize 57
- MKDIR 96
- Monad 196
- move files 92
- mpr 134
- MS Deploy Action 230

MSBuild 158, 163  
 msbuild.exe 84, 162  
 Mscorcfg.msc 180  
 Mscordmp.exe 180  
 msdev 18, 155  
 MSI 73, 143  
 MSI Factory 143  
 msixexec 143  
 MSISStudio 140  
 MSISStudio Action 140  
 MSTest 176, 236  
 MSXML 51, 111, 304, 308  
 MSXML2.DOMDocument 304  
 mta 326, 328  
 Multilizer 149  
 multiple edit 337  
 multiple files 99  
 multiple move 337  
 multiple selection 337

## - N -

Name Property (Macro) 379  
 Name Property (Macros) 382  
 Name Property (Options) 382  
 Name Property (Project) 399  
 Name Property (Scripts) 382  
 Name Property (Step) 411  
 Name Property (Steps) 382  
 NAnt 240, 322  
 navigation 337  
 NBehave 236  
 NCover 241  
 NDepend 243  
 NDoc 243, 322  
 nest 41  
 NestBuildRules Property (Options) 392  
 nested rules 41  
 NestIncludeInBuild Property (Options) 392  
 net use 210  
 network 210, 318  
 Network.bld 318  
 new features 2, 3, 4, 6, 9, 11  
 New Method (Project) 403  
 new step 25  
 newline 43  
 News 318  
 Newsgroup Post Action 210

Ngen.exe 180  
 NMake 180  
 Nmake.exe 180  
 NMAKE\_NET 180  
 NNTP 210, 318  
 NNTPS 210, 211  
 NoLogging Property (Step) 412  
 nooutput 326  
 nsi 141  
 NSIS 73, 141  
 NSIS Action 141  
 NuGet 188  
 NullSoft 141  
 NUnit 236, 245, 322

## - O -

object model 346, 347, 354, 379, 381, 385, 399, 404, 407, 416, 419  
 ObjectModel Sample 324  
 object-selectors 254  
 obsolete 309  
 ocx 219  
 ODBC 217  
 Office 319  
 olb 219  
 OLE 346  
 OLEDB 217  
 Olson Software 127  
 options 41, 42, 43, 44, 57, 58, 59, 60, 61, 255  
 Options Object 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398  
 Options Property (Application) 348  
 Oracle 225  
 Orcas 158  
 order 421  
 OSQL 226  
 output 30, 37, 49  
 output options 60  
 output pane 30, 60  
 override 299  
 overview 1, 23

## - P -

P4 266  
 P4\_CHGLIST 266, 268  
 PackageForTheWeb 136

- PAExec 393
- paint theme 47
- panes 25, 26, 29, 30, 33
- parallel builds 200, 317
- Parallels 291
- parameters 344
- Parameters Property (Macro) 380
- Parent Property (Project) 399
- partial menu 47
- password 45, 47, 49, 66
- Password Property (Project) 400
- pathnames 254
- pause 34, 54, 200, 308, 319
- pause build 34
- Pause Method (Builder) 365
- PCT 204, 209, 211, 214, 216
- percent sign 344
- Perforce 266, 320, 321
- Perforce Action 266
- Perforce.bld 321
- performance 421
- PerlScript 41, 51, 304, 306
- Permview.exe 180
- PersistBuildStatus Property (Options) 392
- PEVerify 189
- Peverify.exe 180
- pfw 136
- piping 64, 66
- PlasticSCM 270
- PlasticSCM Action 270
- play sound 195
- Play Sound Action 195
- Player 295
- Plink 212
- POP 212
- POPS 212
- position 33
- power off 200
- PowerShell 196
- pre-command 66
- pretty print 112
- prevent modification 45
- PreventDuplicateEventLog Property (Options) 393
- preview 49
- print 49
- print options 60
- priority 66, 199, 360
- Prism 158
- process 64
- process files 99
- Process Files Action 99, 319
- process folder 98, 99
- process ID 66, 71, 369
- processes 195, 199
- ProcessID Property (Builder) 360
- PROFILES\_COUNT 99
- PROFILES\_CURRENT 99
- PROFILES\_FILE\_DIR 99
- PROFILES\_FILENAME 99
- PROFILES\_FULLPATH 99
- PROFILES\_ROOT\_DIR 99
- ProcMask 99
- Product Code 161
- profile 24, 35, 53, 326, 328
- ProgID 40
- proj 82
- PROJDIR 340
- project comments 50
- project log file 50
- project log files 344
- Project Object 399, 400, 401, 402, 403, 419
- project options 57, 61
- project properties 50
- Project Property (Application) 348
- project steps 27
- ProjectModified Event (Application) 353
- projects 26
- ProjectSaved Event (Application) 353
- PROJFILE 340
- PROJROOT 340
- prompt 318, 324, 344
- prompt for input 318
- Prompt.bld 318
- PromptMacroValue Event (Application) 353
- properties 30, 50, 57, 221
- Properties Property (Step) 414
- Property Property (Step) 412
- Property2 Property (Step) 413
- PropertyEx Property (Step) 413
- protocol 204, 209, 211, 214, 216
- ps1 196
- PsExec 68, 393
- PsExecCmd (Options) 393
- publish 159
- purchase 421
- PureCM 270

PureCM Action 270  
 put file 201, 206  
 Putty 212  
 PVCS 272, 321  
 PVCS Action 272  
 PVCS.bld 321  
 pvkimprt 110  
 pwd 326, 328  
 Python 51  
 PythonScript 304

## - Q -

query 217, 320  
 QuickInstall 130  
 quote string 308  
 QUOTE\_STR 340  
 quotes 43, 387

## - R -

RAD Studio 82  
 RAR 73  
 Rational 253, 285  
 rationale 1  
 RC4 45  
 rclient 320  
 RD 96  
 read file 307  
 Read File Action 100  
 Read INI Action 101  
 Read Registry Action 198  
 Read XML Action 102  
 read XML file 320  
 READ\_FILE\_VALUE 101  
 READ\_INI 340  
 READ\_INI\_VALUE 101  
 READ\_REGISTRY\_VALUE 198  
 READ\_XML 340  
 READ\_XML\_VALUE 103  
 readonly 107  
 rebuild 34  
 rebuild selected 34  
 recordset 217  
 recurse 99  
 Recurse.bld 319  
 redirection 64, 66  
 redo 338  
 ReevaluateAllBuildRules Property (Options) 393  
 references 181  
 refresh 58  
 REG\_READ 340  
 RegAsm 219  
 REGEDIT 20  
 RegEdit.bld 319  
 regex 343  
 regions 33  
 register 219, 421  
 Register .NET Assembly 219  
 Register .NET Service 184  
 register user action 219  
 RegisterKey Method (Builder) 366  
 registration 303  
 registry 198, 201, 324, 340  
 RegRead 340  
 RegSvcs 184  
 regsvr32 219  
 regular expressions 343  
 Related Items Pane 33  
 relative paths 41, 199  
 reload 57  
 remote 68, 199, 320, 324, 325  
 remove 45  
 remove directory 96  
 Remove Method (Macros) 384  
 Remove Method (Scripts) 406  
 Remove Method (Steps) 419  
 rename files 104  
 Rename Files Action 104  
 RENAMEFILES\_COUNT 104  
 RENAMEFILES\_PROC\_COUNT 104  
 repeat 64  
 repeating 53  
 repeating builds 320  
 replace 47, 105  
 Replace in File Action 105  
 REPLACEINFILE\_MATCH\_COUNT 105  
 REPLACEINFILE\_REPLACE\_COUNT 105  
 reposition 33  
 reset 34, 45  
 reset panes 33  
 ResetBuildStatus Method (Builder) 366  
 ResGen 182  
 resize 33  
 resource files 155, 178



response file 326, 328  
Resume Method (Builder) 366  
retry 54  
return 63  
revisions 422  
right drag 336  
right-click 46  
RMDIR 96  
Robocopy 91  
RoboHelp 128  
RoboHelp Action 128  
roles 222, 223  
root step 57  
RPM 73  
rptproj 158  
RSH 214  
RSpec 236  
RSS 111  
RSS.xslt 112  
RubyScript 51, 304  
rule 43, 53  
RuleCompareTo Property (Step) 415  
RuleComparison Property (Step) 414  
RuleExpression Property (Step) 415  
RuleRepeat Property (Step) 415  
run 64  
run application 64  
run batch file 193  
Run Oracle Script action 225  
run program 64  
Run Program Action 64  
Run Script Action 69, 317, 318  
Run SQL Action 226  
RunProgram Method (Builder) 367  
RUNPROGRAM\_EXITCODE 64, 66, 71, 369  
RUNPROGRAM\_PROCESSID 64, 66, 68, 71, 369  
RunProgram2 Method (Builder) 368  
RunProgramEx Method (Builder) 369  
RunProgramEx2 Method (Builder) 371  
RunProgramInputWaitTimeout Property (Options) 393  
RunProgramOutputBufferSize Property (Options) 393  
RunProgramOutputWaitTimeout Property (Options) 394  
RunProgramRedirInputChunkSize Property (Options) 394  
runs program 40

## - S -

samples 18, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325  
Sandcastle 247  
Save Method (Macros) 384  
Save Method (Options) 398  
Save Method (Project) 402  
Save Method (Scripts) 384  
Save Method (Steps) 384  
SaveStatus.bld 323  
sawvcmd.exe 276  
scheduled builds 329  
Scheduled Tasks 71, 329  
schema 192  
SCM Anywhere 273  
SCM Anywhere Action 273  
scmscmd.exe 273  
screen tips 47  
script 51, 304, 305, 318, 319, 320, 346  
script debugging 304  
script editor 51  
script events 305  
script expressions 304  
Script Object 404  
Script Property (Step) 416  
script user action 322  
Script.bld 319  
scripting 304  
Scripting.Dictionary 304  
scripts 51, 53, 69, 304, 306  
Scripts Collection 404, 405, 406  
Scripts Property (Application) 349  
Scripts Property (Project) 400  
ScriptTypeLibraries Property (Options) 394  
scroll 59  
Seapine 284  
search 47, 105  
secure 204, 209, 211, 212, 214, 216  
security 204, 209, 211, 212, 214, 216  
Secutil.exe 180  
select 337  
selection 48, 52, 337  
self-extracting executable 78  
Send Mail 212, 318  
Send Mail Action 212  
send news post 210

- Serena 262, 272
- server 320
- Server sample 320
- Server.bld 320
- service 228, 320
- Service Action 228
- SERVICE\_START\_TYPE 228
- SERVICE\_STATUS 228
- services 228
- Set Current Dir Action 199
- set current directory 199
- Set File Attributes Action 107
- set macro 70
- Set Macro Action 70, 317, 318
- set version 155
- setenv.bat 66
- SETFILEATTR\_COUNT 107
- SETFILEATTR\_PROC\_COUNT 107
- SetProjectDirectory Property (Options) 394
- Setreg.exe 180
- Setup Factory 143
- Setup Factory Action 143
- Setup Factory for Windows Installer 143
- SetupBuilder 142
- SetupBuilder Action 142
- SFTP 201, 204, 212, 318
- SFX 78
- sh4 123
- sh5 123
- sh6 123
- sh7 123
- share 308
- shell 27, 30, 48, 55, 194, 336, 337
- shell completion 61
- Shift key 46, 337, 338
- shortcut 24, 336
- shortcuts 46
- shut down 200
- sign code 108
- Sign Code action 108
- sign executables 108
- signatures 108
- signcode 108
- Signcode.exe 180
- SIGNCODE\_RESULT 108
- signtool 108
- silent 326
- silent builds (no GUI) 329
- single quotes 387
- SingleInstance.bld 320
- single-step 34
- Sisulizer 150
- skipped 34
- skipped steps 41, 53
- Sleep 200, 308
- Sleep Method (WScript) 420
- sln 158
- Smart Device project 158, 178
- SMTP 212, 318
- SMTSPS 212, 214
- SN 190
- SOAP 320
- Soapsuds.exe 180
- soscmd.exe 278
- sound 59, 195
- SourceAnywhere 276
- SourceAnywhere Action 276
- SourceGear 278, 287
- SourceOffSite 278
- SourceOffSite Action 278
- SourceSafe 166, 320, 322
- SourceSafe Action 166
- SourceSafe.bld 322
- special characters 344
- splitter 33
- SQL 217, 226, 320
- SQL Server 217, 226, 320
- SQL\*Plus 225
- SQLCMD 226
- square brackets 344
- ss.exe 166
- SSH 212, 214, 318
- SSH tunnel 212
- SSH2 212, 214
- SSHHELPER 21
- SSL 204, 209, 211, 214, 216
- stack 32, 308
- standard input 64
- standard output 64
- StarBase 86
- start build 34
- Start In 64
- Start Method (Builder) 371
- StarTeam 86, 321
- StarTeam Action 86
- StarTeam.bld 321

StartEx Method (Builder) 372  
StartEx2 Method (Builder) 372  
StartTime Property (Builder) 360  
statistics 50  
status 34, 340  
Status Property (Builder) 361  
stcmd 86  
stcmd.exe 86  
step 25, 52  
step count 50  
step failure 54  
Step Object 407, 408, 409, 410, 411, 412, 413, 414, 415, 416  
step panes 26  
step properties 52  
StepDone Event 305  
StepDone Event (Builder) 378  
steps 25, 27, 28, 48, 52  
Steps Collection 416, 417, 418, 419  
Steps Property (Application) 350  
Steps Property (Project) 400  
StepStarted Event 305  
StepStarted Event (Builder) 377  
StepStarting Event 305  
StepStarting Event (Builder) 377  
StepType Property (Builder) 361  
stop 34, 63  
stop build 34  
Stop Method (Builder) 374  
Storeadm.exe 180  
string 409  
StripLogLinefeedChar Property (Options) 395  
Strong Name Tool 190  
stylesheets 111  
subroutine 63, 70  
Subroutine Call Action 70, 317  
subroutine steps 28  
subroutines 28  
subst 210  
substitute drive 210  
Subversion 280, 320  
Subversion Action 280, 322  
Subversion.bld 322  
support 422  
Surround SCM 284, 320, 322  
Surround SCM Action 284  
Surround SCM.bld 322  
SVN 280

SyncBuild Method (Builder) 374  
SyncBuildEx Method (Builder) 375  
synchronize folders 92  
syntax highlighting 51  
Sysnative 333  
system 107  
system macros 340  
system requirements 14  
system scripts 306  
system tray 57  
System32 333  
SysWOW64 333

## - T -

tabs 26  
TAR files 73, 78, 79  
target 254  
Tarm QuickInstall 130  
Tarma 130  
Tarma ExpertInstall 130  
Tarma Installer 130  
Tarma InstallMate 130  
task 329  
Task Scheduler 329  
tbuild 168  
TDD 176, 233, 236, 239, 240, 241, 243, 245, 247, 320, 322  
TDD.bld 322  
Team Build 168  
Team Build Action 168  
Team Concert 285  
Team Concert Action 285  
Team Explorer 170  
Team Foundation 170, 320, 323  
Team Foundation Action 170  
Team Foundation Build 168, 332  
Team System 168, 170, 176  
Team System.bld 323  
Team Test 176  
Team Test Action 176  
TEAMBUILD\_BUILDID 168  
TeamCity 332  
Telnet 214, 318  
Telnet Action 214  
TELNET\_HOST\_FINGERPRINT 214  
TEMP 340  
terminate 41, 54

terms 34  
test 55  
Test Driven Development 233, 236, 239, 240, 241, 243, 245, 247, 322  
test step 52  
test-driven development 320  
TestVFP.bld 324  
text log files 42  
TextLoggerCacheMessages Property (Options) 395  
TextLoggerCacheSeconds Property (Options) 396  
TextLogSkippedSteps Property (Options) 395  
TextLogStepEvents Property (Options) 395  
tfsbuild.exe 168  
theme 47  
threading 421  
time 306  
timeout 54  
timestamp 59, 108  
tips 344  
tlb 219  
TlbExp 181  
TlbImp 183  
TLS 204, 209, 211, 214, 216  
tool tips 58, 61  
toolbar 45  
toolbar style 47  
toolbars 24, 45  
TOOLS DIR 340  
toolsfactory 116  
tooltips 47, 338  
touch 107  
transfer files 201, 206  
transform 111  
Transform XML Log Action 111  
TransformLog.xslt 112  
translation 149, 150  
tray 57  
tray icon 57  
TRIM\_QUOTES 340  
Turbo C# 82  
Turbo C++ 82  
Turbo Delphi 82  
Turbo Delphi .NET 82  
type library 219  
Type Property (Macros) 381  
Type Property (Scripts) 405  
Type Property (Steps) 416

## - U -

UCM 253  
underline 58, 61  
undo 338  
undock 33  
unescape 344  
unhide 33  
Uninitialize Method (Application) 352  
Uninitialize Method (Builder) 376  
UniqueEncryptionIV Property (Options) 396  
unit tests 176  
unregister 219  
UNZIP files 74, 78  
UNZIP\_PROCESS\_COUNT 74, 78  
updates 57  
upgrading 16, 18, 20, 21, 22  
UseLogonCreateProcessAsUser Property (Options) 396  
UserProfileOptionWithSyncBuildEx Property (Options) 396  
user action GUI 39  
user actions 38, 301, 322  
User Actions.bld 301  
user configuration 34  
user input 318  
user options 57, 58, 59, 60, 61  
user-defined actions 301  
USERNAME 66, 340  
UseUSEngishLocaleForScriptEngine Property (Options) 397  
UseUTF8ForConsoleApps Property (Options) 397  
UTF-8 44, 397

## - V -

Value Property (Macro) 379  
Value Property (Script) 404  
Vault 287, 320  
Vault Action 287, 322  
Vault.bld 322  
VB 18  
VB.NET 158, 180  
VB.NET actions 301  
VB.NET 20  
VB6 151  
vbBoolean 409

- Vbc.exe 180
- VBCClient 324
- vbld\_AddDelimValue() 308
- vbld\_AllMacros() 308
- vbld\_AppIsRunning() 308
- vbld\_BuildDone 305
- vbld\_BuildStarting 305
- vbld\_BuildStep 40
- vbld\_CompareFileDates() 307
- vbld\_CopyFile() 307
- vbld\_EscapeString() 308
- vbld\_FileDateModified() 307
- vbld\_FileOutOfDate() 307
- vbld\_FindProcess() 308
- vbld\_FormatDate() 306
- vbld\_FormatDateD() 306
- vbld\_FormatDateEx() 306
- vbld\_FormatDateTime() 306
- vbld\_FormatDateD() 306
- vbld\_FormatTime() 306
- vbld\_FormatTimeT() 306
- vbld\_FSO() 307
- vbld\_GetFileContents() 307
- vbld\_If() 308
- vbld\_MakeFileWriteable() 307
- vbld\_MSXML() 308
- vbld\_NextDelimValue() 308
- vbld\_PadLeft() 308
- vbld\_ParseFormattedDateTime() 306
- vbld\_PopDelimValue() 308
- vbld\_ProjectLoaded 305
- vbld\_PushDelimValue() 308
- vbld\_QuoteStr() 308
- vbld\_StepDone 305
- vbld\_StepDoneProject 305
- vbld\_StepProp() 308
- vbld\_StepStarted 305
- vbld\_StepStartedProject 305
- vbld\_StepStarting 305
- vbld\_StepStartingProject 305
- vbld\_TempMacro() 308
- vbld\_TempMacroObj() 308
- vbld\_TempMacros() 308
- vbldBuildCompAborted 358
- vbldBuildCompDone 358
- vbldBuildCompFailed 358
- vbldBuildStatAborting 361
- vbldBuildStatDone 361
- vbldBuildStatPaused 361
- vbldBuildStatPauseReq 361
- vbldBuildStatStarted 361
- vbldContextAutomation 348, 352
- vbldContextCommandLine 348, 352
- vbldContextGUI 348, 352
- vbldLaunchBuild 359
- vbldLaunchNone 359
- vbldLaunchRebuild 359
- vbldLaunchRebuildSel 359
- vbldLogLevelDetailed 390
- vbldLogLevelDiagnostic 390
- vbldLogLevelError 390
- vbldLogLevelNone 390
- vbldLogLevelNormal 390
- vbldLogLevelWarning 390
- vbldMacroAll 349, 381
- vbldMacroGlobal 349, 381
- vbldMacroProject 349, 381
- vbldMacroSystem 349, 381
- vbldMacroTemporary 349, 381
- vbldOutputFile 367, 369
- vbldOutputNone 367, 369
- vbldOutputStdout 367, 369
- vbldRuleContains 414
- vbldRuleDefined 414
- vbldRuleDoesNotContain 414
- vbldRuleEqual 414
- vbldRuleNone 414
- vbldRuleNotEqual 414
- vbldRuleTrue 414
- vbldRuleUndefined 414
- vbldSaveStatusExclude 402
- vbldSaveStatusInclude 402
- vbldSaveStatusUseDefault 402
- vbldScriptAll 349, 405
- vbldScriptGlobal 349, 405
- vbldScriptProject 349, 405
- vbldScriptSystem 349, 405
- vbldScriptTemporary 349, 405
- vbldStepFailure 400, 416
- vbldStepGlobalSubroutine 400, 416
- vbldStepMain 400, 416
- vbldStepStatAborted 367, 369, 407
- vbldStepStatFailed 367, 369, 407
- vbldStepStatInProgress 407
- vbldStepStatMacroError 407
- vbldStepStatPartial 407

- vblStepStatSkipped 407
  - vblStepStatSucceeded 367, 369, 407
  - vblStepSubroutine 400, 416
  - vbLong 409
  - VBNetClient 324
  - VBPLogger sample 325
  - vbproj 158
  - VBScript 41, 51, 304, 306
  - vbString 409
  - VC++ 158
  - VC6 155
  - VCBuild 158
  - vcbuild.exe 162
  - vcp 155
  - vcproj 158
  - vcw 155
  - vdproj 158
  - verify assembly 189
  - version 340
  - versions 38, 178
  - view 30, 48, 55, 57, 337
  - view path 254
  - viewer 57
  - views 33
  - virtual directory 223, 224, 320
  - virtual machines 290, 291, 293, 294, 295, 297
  - Virtual PC 291, 293
  - Virtual Server 293
  - VirtualBox 294
  - VIRTUALSERVER\_VM\_STATE 293
  - VISBUILD 340
  - VISBUILD\_CONFIG\_DIR 340
  - VisBuildCmd 326
  - VISBUILDDIR 340
  - VisBuildPro 328
  - VisBuildPro Project Action 71, 317
  - VisBuildPro sample 324
  - VisBuildPro.bld 324
  - VisBuildPro.config 334
  - VisBuildPro.Global.scripts 334
  - VisBuildPro.macros 334
  - VisBuildPro.steps 334
  - VISBUILDVER 340
  - Visual Basic 151, 178, 323
  - Visual Basic .NET 158, 178
  - Visual C# 158, 178
  - Visual C++ 155, 178, 323
  - Visual C++ .NET 158, 178
  - Visual FoxPro 324
  - Visual J# 178
  - Visual SourceSafe 166, 322
  - Visual Studio 151, 322
  - Visual Studio .NET 151, 158, 178
  - Visual Studio 2005 158, 163
  - Visual Studio 2008 158
  - Visual Studio 2012 158
  - Visual Studio 2013 158
  - Visual Studio 2015 158
  - Visual Studio 6.0 151, 155, 180
  - Visual Studio integration 179
  - Visual Studio sample 322, 323
  - Visual Studio Team System 168, 170, 176
  - Vizacc 123
  - vmrun 295
  - VMware 294, 295
  - VMware ESX Server 294, 295
  - VMware ESXi 295
  - VMware GSX Server 294
  - VMware Player 295
  - VMware Server 294, 295
  - VMware vCenter Server 295
  - VMware vSphere 295
  - VMware Workstation 295
  - VMWARESERVER\_VM\_STATE 294
  - Votive 158
  - VS 2005 158
  - VS.NET 158
  - VS.NET Get Version Action 178
  - VS6 Get Version Action 178
  - VS6\_PROJ\_VER 178
  - VSMake 16
  - VSNET\_ASSEM\_VER 178
  - VSNET\_PROJ\_VER 178
  - vSphere 295
  - VSS 166
  - VSTS 168, 170, 176, 323
  - VStudio.bld 323
  - vsvars32.bat 66
  - VT\_BOOL 409
  - VT\_BSTR 409
  - VT\_I4 409
- W -**
- wait 66, 200
  - Wait Action 200, 317

watches 32  
WBEM 308  
WDK 66  
wdproj 158  
Web Deploy 230  
web interface 324  
web page 318  
web service 320  
web services 190  
Web Services Description Language 190  
WebLauncher 324  
Whidbey 158  
why 1  
wildcards 339  
WIM 73  
Win CE 158, 178  
Wincv.exe 180  
WINDIR 340  
windows 33  
Windows Installer 143  
Windows Installer XML 146  
Windows Virtual PC 297  
WINSYSDIR 340  
WinZip 18, 319  
Wise Action 145  
Wise Clean Build 146  
Wise for Windows Installer 145  
Wise Installation Studio 145  
Wise Installation System 145  
Wise Package Studio 145  
Wise Solutions 145  
wltem Installer 136  
WiX 146, 158  
wixproj 158  
WMI 195, 200, 308  
working folder 199  
WoW64 333  
Write File Action 114, 317  
Write INI Action 114  
Write Registry Action 201, 319  
Write XML Action 115  
write XML file 320  
WriteBOM Property (Options) 397  
WScript Object 419, 420  
WScript.Shell 304  
WSDL 190  
WshShell 304

## - X -

x64 333  
x86 333  
xcopy 91  
XML 111, 192, 320, 335, 340  
XML files 102, 115  
XML log files 42  
XML logging 111  
XML Scheme Definition 192  
XML.bld 320  
XMLLoggerCacheMessages Property (Options) 398  
XMLLoggerCacheSeconds Property (Options) 397  
XP 233, 239, 240, 241, 243, 245, 247, 322  
XP Mode 297  
xpj 128  
XSD 192  
XSL 111  
XSLT 111  
xUnit.Net 236

## - Z -

Z 73  
ZeroG 133, 134  
ZIP files 73, 76, 79  
ZIP\_PROCESS\_COUNT 76, 79