Visual Smalltalk Enterprise™™

Part Reference

P46-0202-00
## Table of Contents

### Introduction
- About This Manual ................................................................. xxi
- How to Read a Part Reference Chapter ..................................... xxi

### Chapter 1 Alphabetic Entry Field
- Events .................................................................................. 25
- Messages .............................................................................. 26
- Direct-Edit ........................................................................... 27
- Property Edit ......................................................................... 28
- Related Parts ....................................................................... 29
- Example ............................................................................... 30

### Chapter 2 Application
- Operation ............................................................................. 31
- Events .................................................................................. 32
- Messages .............................................................................. 32
- Direct-Edit ........................................................................... 32
- Property Edit ......................................................................... 33
- Related Parts ....................................................................... 34
- Remarks ............................................................................... 34
- Example ............................................................................... 35

### Chapter 3 Array Holder
- Operation ............................................................................... 37
- Events .................................................................................. 37
- Messages .............................................................................. 38
- Direct-Edit ........................................................................... 40
- Property Edit ......................................................................... 40
- Related Parts ....................................................................... 42
- Remarks ............................................................................... 42

### Chapter 4 AVI Video (Win32 only)
- Operation ............................................................................... 43
- Events .................................................................................. 43
- Messages .............................................................................. 43
- Direct-Edit ........................................................................... 47
- Property Edit ......................................................................... 47
- Related Parts ....................................................................... 47
- Example ............................................................................... 48
## Chapter 5 Btrieve Accessor 49
- Operation ................................................................. 49
- Events ....................................................................... 51
- Messages ................................................................. 51
- Direct-Edit .............................................................. 54
- Property Edit .......................................................... 54
- Setting Field Properties .......................................... 57
- Remarks ................................................................. 58
- Example ................................................................. 62

## Chapter 6 Button List Box 63
- Operation ................................................................. 63
- Events ....................................................................... 63
- Messages ................................................................. 65
- Direct-Edit .............................................................. 67
- Property Edit .......................................................... 67
- Related Parts .......................................................... 68
- Example ................................................................. 69

## Chapter 7 C Structure 71
- Operation ................................................................. 71
- Events ....................................................................... 73
- Messages ................................................................. 74
- Direct-Edit .............................................................. 74
- Property Edit .......................................................... 75
- Related Parts .......................................................... 77
- Remarks ................................................................. 77

## Chapter 8 CD Audio (Win32 only) 79
- Operation ................................................................. 79
- Events ....................................................................... 79
- Messages ................................................................. 79
- Direct-Edit .............................................................. 83
- Property Edit .......................................................... 83
- Related Parts .......................................................... 84
- Example ................................................................. 84

## Chapter 9 Check Box 85
- Operation ................................................................. 85
- Events ....................................................................... 86
- Messages ................................................................. 86
- Direct-Edit .............................................................. 87
- Property Edit .......................................................... 88
- Related Parts .......................................................... 89
- Remarks ................................................................. 89
- Example ................................................................. 89

## Chapter 10 Clipboard Accessor 91
- Events ....................................................................... 91
- Messages ................................................................. 91
- Direct-Edit .............................................................. 92
<table>
<thead>
<tr>
<th>Chapter 11</th>
<th>Combo Box</th>
<th>95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>Remarks</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 12</th>
<th>Comparison</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>103</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 13</th>
<th>Computation</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>107</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 14</th>
<th>Confirmer</th>
<th>109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>112</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 15</th>
<th>Confirmer with Cancel</th>
<th>113</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>116</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 16</th>
<th>Container (OS/2 only)</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>Chapter</td>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td><strong>Chapter 17</strong></td>
<td>Conversion</td>
<td>135</td>
</tr>
<tr>
<td>Remarks</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 18</strong></td>
<td>Currency Entry Field</td>
<td>139</td>
</tr>
<tr>
<td>Events</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 19</strong></td>
<td>Date Entry Field</td>
<td>145</td>
</tr>
<tr>
<td>Events</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>148</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 20</strong></td>
<td>DDE Client</td>
<td>151</td>
</tr>
<tr>
<td>Operation</td>
<td>151</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>Remarks</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 21</strong></td>
<td>DDE Server</td>
<td>157</td>
</tr>
<tr>
<td>Operation</td>
<td>157</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Remarks</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 22</strong></td>
<td>Dial Pane</td>
<td>161</td>
</tr>
<tr>
<td>Operation</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>162</td>
<td></td>
</tr>
<tr>
<td>Chapter 23</td>
<td>Dialog Window</td>
<td>167</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>-----</td>
</tr>
<tr>
<td>Operation</td>
<td>...............................</td>
<td>167</td>
</tr>
<tr>
<td>Events</td>
<td>..................................</td>
<td>170</td>
</tr>
<tr>
<td>Messages</td>
<td>..................................</td>
<td>170</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>..................................</td>
<td>171</td>
</tr>
<tr>
<td>Property Edit</td>
<td>..................................</td>
<td>171</td>
</tr>
<tr>
<td>Related Parts</td>
<td>..................................</td>
<td>173</td>
</tr>
<tr>
<td>Example</td>
<td>..................................</td>
<td>174</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 24</th>
<th>Dictionary Holder</th>
<th>175</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>..................................</td>
<td>175</td>
</tr>
<tr>
<td>Events</td>
<td>..................................</td>
<td>175</td>
</tr>
<tr>
<td>Messages</td>
<td>..................................</td>
<td>176</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>..................................</td>
<td>178</td>
</tr>
<tr>
<td>Property Edit</td>
<td>..................................</td>
<td>178</td>
</tr>
<tr>
<td>Remarks</td>
<td>..................................</td>
<td>180</td>
</tr>
<tr>
<td>Example</td>
<td>..................................</td>
<td>180</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 25</th>
<th>Disk Accessor</th>
<th>181</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
<td>...............................</td>
<td>181</td>
</tr>
<tr>
<td>Messages</td>
<td>..................................</td>
<td>182</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>..................................</td>
<td>185</td>
</tr>
<tr>
<td>Property Edit</td>
<td>..................................</td>
<td>185</td>
</tr>
<tr>
<td>Remarks</td>
<td>..................................</td>
<td>185</td>
</tr>
<tr>
<td>Example</td>
<td>..................................</td>
<td>186</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 26</th>
<th>DLL Accessor</th>
<th>187</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>...............................</td>
<td>188</td>
</tr>
<tr>
<td>Events</td>
<td>..................................</td>
<td>188</td>
</tr>
<tr>
<td>Messages</td>
<td>..................................</td>
<td>188</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>..................................</td>
<td>188</td>
</tr>
<tr>
<td>Property Edit</td>
<td>..................................</td>
<td>189</td>
</tr>
<tr>
<td>Related Parts</td>
<td>..................................</td>
<td>191</td>
</tr>
<tr>
<td>Remarks</td>
<td>..................................</td>
<td>192</td>
</tr>
<tr>
<td>Example for Windows</td>
<td>..................................</td>
<td>192</td>
</tr>
<tr>
<td>Example for OS/2</td>
<td>..................................</td>
<td>193</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 27</th>
<th>Drawn Button</th>
<th>195</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>...............................</td>
<td>195</td>
</tr>
<tr>
<td>Events</td>
<td>..................................</td>
<td>195</td>
</tr>
<tr>
<td>Messages</td>
<td>..................................</td>
<td>196</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>..................................</td>
<td>196</td>
</tr>
<tr>
<td>Property Edit</td>
<td>..................................</td>
<td>197</td>
</tr>
<tr>
<td>Related Parts</td>
<td>..................................</td>
<td>197</td>
</tr>
<tr>
<td>Example</td>
<td>..................................</td>
<td>197</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Pages</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>28</td>
<td>Drop-Down List</td>
<td>199</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>199</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>203</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>204</td>
</tr>
<tr>
<td>29</td>
<td>Edit Menu</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>208</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>208</td>
</tr>
<tr>
<td>30</td>
<td>Entry Field</td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>209</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>211</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>212</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>213</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>214</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>215</td>
</tr>
<tr>
<td>31</td>
<td>File Accessor</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>220</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>221</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>221</td>
</tr>
<tr>
<td>32</td>
<td>File Dialog</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>223</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>225</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>224</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>225</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>225</td>
</tr>
<tr>
<td>33</td>
<td>File Menu</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>227</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>228</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>228</td>
</tr>
<tr>
<td>Chapter 34</td>
<td>Float Entry Field</td>
<td>229</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Events</td>
<td>229</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>230</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>232</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>233</td>
<td></td>
</tr>
<tr>
<td>Chapter 35</td>
<td>Format Menu (Win32 only)</td>
<td>235</td>
</tr>
<tr>
<td>Operation</td>
<td>235</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>236</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>Chapter 36</td>
<td>Graph Pane</td>
<td>239</td>
</tr>
<tr>
<td>Events</td>
<td>239</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>Chapter 37</td>
<td>Group Pane</td>
<td>243</td>
</tr>
<tr>
<td>Events</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>Chapter 38</td>
<td>Header Control (Win32 only)</td>
<td>245</td>
</tr>
<tr>
<td>Operation</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>245</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>246</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>247</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>248</td>
<td></td>
</tr>
<tr>
<td>Chapter 39</td>
<td>Help Menu</td>
<td>249</td>
</tr>
<tr>
<td>Operation</td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Property Edit</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Related Parts</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Example</td>
<td>253</td>
<td></td>
</tr>
</tbody>
</table>
## Table of Contents

### Chapter 40 Information Dialog
- Operation ................................................................. 255
- Events ........................................................................ 255
- Messages ................................................................. 256
- Direct-Edit ............................................................... 257
- Property Edit ............................................................ 257
- Example ..................................................................... 258

### Chapter 41 Integer Entry Field
- Events ......................................................................... 259
- Messages ..................................................................... 260
- Direct-Edit ............................................................... 262
- Property Edit ............................................................ 262
- Related Parts ............................................................ 263
- Example .................................................................... 263

### Chapter 42 Launch Pad
- Operation ................................................................. 265
- Events ......................................................................... 266
- Messages ..................................................................... 266
- Direct-Edit ............................................................... 266
- Property Edit ............................................................ 266
- Example .................................................................... 267

### Chapter 43 Link Junction
- Operation ................................................................. 269
- Events ......................................................................... 271
- Messages ..................................................................... 271
- Direct-Edit ............................................................... 272
- Property Edit ............................................................ 272
- Remarks ..................................................................... 273
- Example .................................................................... 273

### Chapter 44 List Pane
- Operation ................................................................. 275
- Events ......................................................................... 276
- Messages ..................................................................... 276
- Direct-Edit ............................................................... 278
- Property Edit ............................................................ 278
- Related Parts ............................................................ 280
- Example .................................................................... 280

### Chapter 45 List View (Win32 only)
- Operation ................................................................. 283
- Events ......................................................................... 284
- Messages ..................................................................... 285
- Direct-Edit ............................................................... 287
- Property Edit ............................................................ 288
- Related Parts ............................................................ 288
- Example .................................................................... 291
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Menu</th>
<th>Operation</th>
<th>Events</th>
<th>Messages</th>
<th>Direct-Edit</th>
<th>Property Edit</th>
<th>Related Parts</th>
<th>Remarks</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>Menu</td>
<td>293</td>
<td>293</td>
<td>294</td>
<td>295</td>
<td>295</td>
<td>295</td>
<td>295</td>
<td>295</td>
</tr>
<tr>
<td>47</td>
<td>Menu Bitmap Item</td>
<td>297</td>
<td>297</td>
<td>298</td>
<td>299</td>
<td>299</td>
<td>299</td>
<td>299</td>
<td>299</td>
</tr>
<tr>
<td>48</td>
<td>Menu Item Separator</td>
<td>301</td>
<td>301</td>
<td>301</td>
<td>302</td>
<td>302</td>
<td>302</td>
<td>302</td>
<td>302</td>
</tr>
<tr>
<td>49</td>
<td>Menu Label</td>
<td>303</td>
<td>303</td>
<td>304</td>
<td>304</td>
<td>304</td>
<td>305</td>
<td>305</td>
<td>305</td>
</tr>
<tr>
<td>50</td>
<td>Menu Text Item</td>
<td>307</td>
<td>307</td>
<td>308</td>
<td>309</td>
<td>309</td>
<td>310</td>
<td>310</td>
<td>310</td>
</tr>
<tr>
<td>51</td>
<td>Multiple Choice List</td>
<td>311</td>
<td>311</td>
<td>312</td>
<td>312</td>
<td>314</td>
<td>315</td>
<td>315</td>
<td>315</td>
</tr>
</tbody>
</table>
Table of Contents

Related Parts ................................................................. 316
Example ................................................................. 317

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Part</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>Nested Part</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>325</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>325</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>326</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>326</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>326</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Part</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>53</td>
<td>Notebook</td>
<td>327</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>327</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>329</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>331</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>331</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>333</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Part</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>54</td>
<td>Notebook Page</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>337</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>337</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>337</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Part</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>Number Holder</td>
<td>339</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>339</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>340</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>340</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>349</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>349</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>350</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Part</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>OLE Control (Win32 only)</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>351</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>352</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>352</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>352</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>353</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Part</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>Ordered Collection Holder</td>
<td>355</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>355</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>355</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>359</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>359</td>
</tr>
<tr>
<td>Chapter</td>
<td>Part Accessor</td>
<td>Phone Number Entry Field</td>
</tr>
<tr>
<td>---------</td>
<td>---------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td></td>
</tr>
</tbody>
</table>
### Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Property Constructor</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Property Edit</td>
<td>393</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>394</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>395</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>396</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>396</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>396</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>396</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>398</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>399</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Property Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operation</td>
<td>403</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>404</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>406</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>406</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>407</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>408</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Property Sheet (Win32 only)</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operation</td>
<td>411</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>412</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>412</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>412</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>412</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>412</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>413</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>413</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Push Button</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operation</td>
<td>419</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>419</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>420</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>421</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>422</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>422</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Radio Button</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Operation</td>
<td>423</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>426</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>427</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>427</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>428</td>
</tr>
<tr>
<td>Chapter 69</td>
<td>Report Writer</td>
<td>429</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>Chapter 70</td>
<td>Rich Edit (Win32 only)</td>
<td>431</td>
</tr>
<tr>
<td>Operation</td>
<td>.................................................................</td>
<td>431</td>
</tr>
<tr>
<td>Events</td>
<td>.................................................................</td>
<td>432</td>
</tr>
<tr>
<td>Messages</td>
<td>.................................................................</td>
<td>433</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>.................................................................</td>
<td>436</td>
</tr>
<tr>
<td>Property Edit</td>
<td>.................................................................</td>
<td>436</td>
</tr>
<tr>
<td>Related Parts</td>
<td>.................................................................</td>
<td>438</td>
</tr>
<tr>
<td>Example</td>
<td>.................................................................</td>
<td>438</td>
</tr>
<tr>
<td>Chapter 71</td>
<td>Scroll Bars (Win32 only)</td>
<td>439</td>
</tr>
<tr>
<td>Operation</td>
<td>.................................................................</td>
<td>439</td>
</tr>
<tr>
<td>Events</td>
<td>.................................................................</td>
<td>439</td>
</tr>
<tr>
<td>Messages</td>
<td>.................................................................</td>
<td>440</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>.................................................................</td>
<td>440</td>
</tr>
<tr>
<td>Property Edit</td>
<td>.................................................................</td>
<td>441</td>
</tr>
<tr>
<td>Related Parts</td>
<td>.................................................................</td>
<td>441</td>
</tr>
<tr>
<td>Example</td>
<td>.................................................................</td>
<td>442</td>
</tr>
<tr>
<td>Chapter 72</td>
<td>Sliders (OS/2 only)</td>
<td>443</td>
</tr>
<tr>
<td>Operation</td>
<td>.................................................................</td>
<td>444</td>
</tr>
<tr>
<td>Events</td>
<td>.................................................................</td>
<td>444</td>
</tr>
<tr>
<td>Messages</td>
<td>.................................................................</td>
<td>444</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>.................................................................</td>
<td>445</td>
</tr>
<tr>
<td>Property Edit</td>
<td>.................................................................</td>
<td>445</td>
</tr>
<tr>
<td>Related Parts</td>
<td>.................................................................</td>
<td>448</td>
</tr>
<tr>
<td>Remarks</td>
<td>.................................................................</td>
<td>448</td>
</tr>
<tr>
<td>Example</td>
<td>.................................................................</td>
<td>448</td>
</tr>
<tr>
<td>Chapter 73</td>
<td>Speaker</td>
<td>449</td>
</tr>
<tr>
<td>Events</td>
<td>.................................................................</td>
<td>449</td>
</tr>
<tr>
<td>Messages</td>
<td>.................................................................</td>
<td>449</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>.................................................................</td>
<td>450</td>
</tr>
<tr>
<td>Property Edit</td>
<td>.................................................................</td>
<td>450</td>
</tr>
<tr>
<td>Remarks</td>
<td>.................................................................</td>
<td>450</td>
</tr>
<tr>
<td>Example</td>
<td>.................................................................</td>
<td>451</td>
</tr>
<tr>
<td>Chapter 74</td>
<td>Spin Buttons (OS/2 only)</td>
<td>453</td>
</tr>
<tr>
<td>Events</td>
<td>.................................................................</td>
<td>453</td>
</tr>
<tr>
<td>Messages</td>
<td>.................................................................</td>
<td>454</td>
</tr>
<tr>
<td>Direct-Edit</td>
<td>.................................................................</td>
<td>455</td>
</tr>
<tr>
<td>Property Edit for Master Spin Buttons</td>
<td>.................................................................</td>
<td>455</td>
</tr>
<tr>
<td>Property Edit for Slave Spin Buttons</td>
<td>.................................................................</td>
<td>458</td>
</tr>
<tr>
<td>Remarks</td>
<td>.................................................................</td>
<td>459</td>
</tr>
<tr>
<td>Example</td>
<td>.................................................................</td>
<td>459</td>
</tr>
<tr>
<td>Chapter</td>
<td>Part Name</td>
<td>Page</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>75</td>
<td>Static Graphic</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>462</td>
</tr>
<tr>
<td>76</td>
<td>Static Text</td>
<td>463</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>463</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>463</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>464</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>464</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>464</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>465</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>465</td>
</tr>
<tr>
<td>77</td>
<td>Status Window (Win32 only)</td>
<td>467</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>467</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>467</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>468</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>468</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>468</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>472</td>
</tr>
<tr>
<td>78</td>
<td>String Holder</td>
<td>473</td>
</tr>
<tr>
<td></td>
<td>Event</td>
<td>473</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>474</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>478</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>478</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>478</td>
</tr>
<tr>
<td>79</td>
<td>String Template</td>
<td>481</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>481</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>481</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>481</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>482</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>482</td>
</tr>
<tr>
<td></td>
<td>Remarks</td>
<td>483</td>
</tr>
<tr>
<td>80</td>
<td>Tab Control (Win32 only)</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>Operation</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>Events</td>
<td>486</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>487</td>
</tr>
<tr>
<td></td>
<td>Direct-Edit</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>Property Edit</td>
<td>489</td>
</tr>
<tr>
<td></td>
<td>Related Parts</td>
<td>492</td>
</tr>
<tr>
<td></td>
<td>Example</td>
<td>492</td>
</tr>
</tbody>
</table>
### Chapter 81  Tab Control Page (Win32 only)

- **Operation** 495
- **Events** 495
- **Messages** 496
- **Direct-Edit** 496
- **Property Edit** 497
- **Related Parts** 498
- **Remarks** 498
- **Example** 499

### Chapter 82  Table Pane

- **Operation** 502
- **Events** 503
- **Messages** 505
- **Direct-Edit** 510
- **Property Edit** 511
- **Related Parts** 513
- **Remarks** 514
- **Example** 517

### Chapter 83  Text Pane

- **Operation** 519
- **Events** 520
- **Messages** 520
- **Direct-Edit** 523
- **Property Edit** 523
- **Related Parts** 525
- **Example** 525

### Chapter 84  Timer

- **Events** 527
- **Messages** 527
- **Direct-Edit** 528
- **Property Edit** 528
- **Remarks** 528
- **Example** 529

### Chapter 85  Tool Box (Win32 only)

- **Operation** 531
- **Events** 531
- **Messages** 531
- **Direct-Edit** 532
- **Property Edit** 532
- **Related Parts** 532
- **Remarks** 532
- **Example** 532
## Table of Contents

### Chapter 86 Toolbar (Win32 only)

- Operation ................................................................. 533
- Events ................................................................. 535
- Messages ............................................................. 536
- Direct-Edit ......................................................... 537
- Property Edit ..................................................... 537
- Related Parts ....................................................... 540
- Remarks ............................................................. 540
- Example ............................................................ 541

### Chapter 87 Toolbar Button (Win32 only)

- Operation ................................................................. 543
- Events ................................................................. 544
- Messages ............................................................. 544
- Direct-Edit ......................................................... 545
- Property Edit ..................................................... 545
- Related Parts ....................................................... 548
- Example ............................................................ 548

### Chapter 88 Toolbar Separator (Win32 only)

- Events ................................................................. 549
- Messages ............................................................. 549
- Direct-Edit ......................................................... 549
- Property Edit ..................................................... 550
- Related Parts ....................................................... 550
- Example ............................................................ 551

### Chapter 89 Toolbar Toggle Button (Win32 only)

- Operation ................................................................. 553
- Events ................................................................. 554
- Messages ............................................................. 554
- Direct-Edit ......................................................... 556
- Property Edit ..................................................... 556
- Related Parts ....................................................... 558
- Example ............................................................ 559

### Chapter 90 Track Bar

- Operation ................................................................. 561
- Events ................................................................. 561
- Messages ............................................................. 562
- Direct-Edit ......................................................... 565
- Property Edit ..................................................... 565
- Related Parts ....................................................... 567
- Example ............................................................ 568

### Chapter 91 Tree View (Win32 only)

- Operation ................................................................. 569
- Events ................................................................. 570
- Messages ............................................................. 571
- Direct-Edit ......................................................... 572
- Property Edit ..................................................... 573
# Table of Contents

Related Parts ............................................................... 574
Example ................................................................. 574

## Chapter 92 Value Holder 577

- Operation ........................................................................................................... 577
- Events ............................................................................................................... 579
- Messages ........................................................................................................... 579
- Direct-Edit ........................................................................................................ 580
- Property Edit .................................................................................................... 580
- Related Parts ................................................................................................. 581
- Example ........................................................................................................... 581

## Chapter 93 Value Set (OS/2 only) 585

- Operation ........................................................................................................... 586
- Events ............................................................................................................... 586
- Messages ........................................................................................................... 587
- Direct-Edit ........................................................................................................ 588
- Property Edit .................................................................................................... 589
- Related Parts ................................................................................................. 590
- Remarks ........................................................................................................... 590
- Example ........................................................................................................... 591

## Chapter 94 Video Pane (Win32 only) 593

- Operation ........................................................................................................... 593
- Events ............................................................................................................... 593
- Messages ........................................................................................................... 593
- Direct-Edit ........................................................................................................ 594
- Property Edit .................................................................................................... 594
- Related Parts ................................................................................................. 595
- Example ........................................................................................................... 596

## Chapter 95 Wave Audio (Win32 only) 597

- Operation ........................................................................................................... 597
- Events ............................................................................................................... 597
- Messages ........................................................................................................... 597
- Direct-Edit ........................................................................................................ 601
- Property Edit .................................................................................................... 601
- Related Parts ................................................................................................. 601
- Example ........................................................................................................... 602

## Chapter 96 Window 603

- Operation ........................................................................................................... 604
- Events ............................................................................................................... 605
- Messages ........................................................................................................... 606
- Direct-Edit ........................................................................................................ 607
- Property Edit .................................................................................................... 607
- Related Parts ................................................................................................. 610
- Example ........................................................................................................... 610
# Table of Contents

## Chapter 97 Wizard (Win32 only) 611
- Operation .......................................................... 611
- Events ............................................................... 612
- Messages ............................................................ 612
- Direct-Edit .......................................................... 612
- Property Edit ....................................................... 612
- Related Parts ...................................................... 613
- Example .............................................................. 613

## Appendix A Event/Part Cross-Reference 615

## Appendix B Message/Part Cross-Reference 619

## Appendix C Standard Part Properties 631
- Backcolor Property ............................................... 632
- Edit bitmap Property ............................................ 634
- Edit icon Property ............................................... 635
- Font Property ...................................................... 638
- Forecolor Property ............................................... 639
- Order children Property ....................................... 640
- Part name Property .............................................. 643
- Size window Property .......................................... 644

## Index 647
Welcome to the PARTS Workbench® subsystem of Visual Smalltalk™. The Parts Assembly and Reuse Tool Set is a visual, object-oriented application development system that lets you quickly construct applications by visually linking pre-fabricated parts on screen.

PARTS Workbench contains two different types of parts: you use visual parts to build the user interface; you use non-visual parts to perform behind-the-scenes operations such as accessing a record in a database.

See the Visual Smalltalk Enterprise Workbench User's Guide to learn more about the PARTS Workbench environment.

About This Manual

This manual is a reference to each of the standard parts provided with the product. Each part is described in a standard format, with on-line examples provided for you to load and exercise.

We use the following general notation conventions in this book:

- Event and message names are printed **boldface** in the standard typeface.
- Menu and menu choice names are separated by a slash when both appear together. For example View/Show links refers to the pull-down menu path activated by first clicking with mouse button 1 on the keyword View in the menu bar and then on the Show links menu choice.

Visual Smalltalk documentation also includes:

- **Getting Started**, which explains how to install Visual Smalltalk (including the PARTS Workbench)
- **Visual Smalltalk Enterprise Workbench User's Guide**, which introduces the PARTS Workbench, and explains how to design, link, debug, execute, and ship your applications
- **Visual Smalltalk Enterprise Tutorial**, which describes how to build applications in the Visual Smalltalk environment, using features of the PARTS Workbench and Smalltalk

How to Read a Part Reference Chapter

Each part reference chapter contains some or all of the following sections, as appropriate:

- Introduction
- Operation
- Events
- Messages
The following paragraphs briefly describe the type of information that each of these sections contains.

**Introduction**

The beginning of each part reference typically includes the following information:

- The icon that appears in the catalog for the part
- The name of the part (and if the part is not available on all operating systems, a parenthetical indication of the operating systems that the part is available on)
- Whether the part is visual or nonvisual
- Its purpose
- How it is typically used
- For a visual part, an illustration indicating its appearance at runtime

**Operation**

Describes usage of the part from both the end-user’s and developer’s point of view.

**Events**

A description of the part’s events which are available in the Create Link dialog. The following illustrates the format used to define events.

**changed:** *aString*

Occurs when the value is changed. The new value is provided by *aString*.

  **Triggered By:** *setValue:*

**clicked**

Occurs when the user clicks on the part.

  **Triggered By:** *the user*

**field:** *aValue*

An automatically generated event.

The first part of the event definition includes the name of the event in bold followed by the event’s values, if any, in italics. In the above example, the **changed:** event has a value of type String. Some event names are both bold and italic such as the **field:** event above. The italics indicate a variable portion of the event name; the actual name of the event is generated by the part. For example, the Btrieve accessor generates a **field:** event for each field in the database. For the field “foo”, the event is **foo:**.

The next part of the definition is a description of the message and its arguments (if any). **Triggered By:** indicates who or what triggers the event. This is either *the user*, when the event is triggered by a user action such as a mouse click, or a message name when the event is triggered by a message.
Messages

A description of the part’s messages which are available in the Create Link dialog. The following illustrates the format used to define part messages.

**enable**

Recognizes user input.

*Returns:* —

*Triggers:* —

**setField: aValue**

This is an automatically generated message.

**setValue: aString**

Sets the part value to the argument.

*Returns:* *aString*

*Triggers:* *changed:*

**value**

Returns the part value.

*Returns:* *a String*

*Triggers:* —

The first entry in the message definition includes the name of the message in bold followed by the message’s arguments (if any) in italics. In the above example, the `setValue:` message requires an argument of type `String`. Some message names are both bold and italic, such as the `setField:` message above. The italics indicate a variable portion of the message name; the actual name of the message is generated by the part. For example, the Btrieve accessor generates a `setField:` message for each field in the database. For the field “foo”, the message is `setfoo:`.

The next part of the definition is a description of the message and its arguments (if any). *Returns:* shows the value (if any) that the message returns. `value` and `setValue:` both return a string, but `value` lists “a String” while the return value of `setValue:` is “*aString*”. The Roman font of “*a String*” indicates that the return value is of `String` type. The description explains what the value represents.

The italic font of “*aString*” indicates that the message argument itself is returned as a side effect of executing the message. Many messages that require an argument also return that argument as a result so the value can be used again in a result link.

*Triggers:* lists the events that can be triggered by the message. Some events are triggered conditionally, so the appearance of an event name doesn’t mean that it is always triggered by the message. If an event is not mentioned in the message’s description, then that event is always triggered by the message. Messages are listed alphabetically.

**Direct-Edit**

This section tells you which property is affected when you direct-edit the part. Direct-edit for nonvisual parts usually edits the part name, which appears as the icon label beneath the part icon in the workbench. Direct-edit for visual parts usually edits the contents or the value of the part.
To direct-edit a part:

> ALT <click> on the part.

or

> Select the part and choose Selected/Edit direct.

**Property Edit**

- **Standard Properties**
  The first property is always the part’s name. Most visual parts also have standard Size window, Font, Forecolor, and Backcolor dialogs. Most nonvisual parts have an Edit icon... button that lets you change the part’s icon. You cannot edit the icons for string and number holders, because they are used to display the holders' contents. See Appendix C, Standard Part Properties, in this document for a description of the standard part properties.

- **Part-Specific Properties**
  This section shows an illustration of each Properties dialog that is specific to this part, along with an explanation of each control.

To access a part’s Properties dialog:

> Double-click on the part.

or

> Select the part and choose the Selected/Edit properties... command.

**Related Parts**

This section describes the relationship between this part and any other parts that are always or frequently used with this part, or whose operation or appearance resemble this part.

**Remarks**

This optional section contains any general or additional information about the part that hasn’t been covered elsewhere, such as:

- Efficiency considerations
- Licensing restrictions
- Usage considerations or conventions
- Defining keyboard shortcuts

**Example**

This optional section describes and illustrates an application that lets you experiment with the part using the most common events and messages. A screen shot of the application's workbench shows the principal links. Each example is included as a PAR file, usually in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. The name of the file and any additional instructions for using the application are included in this section.
Alphabetic entry field parts are operated just like entry field parts, except that the characters that you can enter are restricted to letters. If you try to enter a non-letter character, the system speaker beeps. You can set the value of an empty alphabetic entry field to be either null or an empty string in the field’s Properties dialog.

The OS/2 version of this part has an additional Alignment property.

Events

aboutToChangeTo: aValue

Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the clear or setValue: messages are executed. This event is triggered only after aValue has been validated by the entry field’s built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the abortChange and retryChange messages to respond to the proposed new value. If this event is triggered by the clear or setValue: messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its changed: event with aValue as its event value.

Triggered By: the user

clear

setValue:

changed: aString

Occurs when the field loses input focus and the text has changed since the input focus entered this field, or when the clear or setValue: messages are executed. The new contents are provided in aString.

Triggered By: the user

clear

setValue:

entered: aString

Occurs when the Enter key is pressed. The contents are provided in aString.

Triggered By: the user
Alphabetic Entry Field

rightClicked
Occurs when the user clicks button 2 in the field.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

valueException: aValue
Triggered when the argument of the setValue: message is not a valid value for the field.

Triggered By: setValue:

Messages

abortChange
Rejects the new value entered by the user and resets the entry field’s contents to the original value. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —

clear
Sets the contents to an empty string or null depending on the setting of the “Value when empty” property in the Properties dialog.

Returns: —
Triggers: changed:

disable
Ignores user input. The text in a disabled alphabetic entry field is dimmed.

Returns: —
Triggers: —

enable
Enables the field to respond to user input.

Returns: —
Triggers: —

retryChange
Rejects the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —
selectAll
Selects all the text in the field.

Returns: —
Triggers: —

setFocus
Moves the input focus to the field. Input focus is indicated by the blinking text cursor.

Returns: —
Triggers: —

setValue: aString
Sets the contents of the field to aString. A warning dialog is displayed and the contents of the field are not changed if aString is not valid input to an alphabetic entry field.

Returns: —
Triggers: changed:
valueException:

value
Returns the contents of the field, a String or null.

Returns: a String
Triggers: —

valueFormatted
Returns the contents of the field.

Returns: a String
Triggers: —

Direct-Edit
Direct-editing an alphabetic entry field changes its contents. An Invalid input dialog is displayed if you enter characters that are not allowed in the field.
Property Edit

The part-specific properties of the alphabetic entry field are described in the following paragraphs. The OS/2 version of this part has an additional Alignment property.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Contents

You can change the contents of the field either by direct-editing or by typing a value in this field.

Maximum number of characters

Specifies the maximum number of characters that a user can enter in the field.

Value when empty

When the field is empty, it can return either an empty string or the special value null.

Styles

- **Border**
  
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**
  
  When checked, the characters in the field can be scrolled left and right if they are not entirely visible within the entry field. If this is not checked, the characters cannot be scrolled. You can continue to enter characters whether or not they are visible in the field. The default is to allow scrolling.
• **Read Only**
  When checked, the contents cannot be changed by typing. The default is to allow changes.

• **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

• **Alignment**
  For the OS/2 version of this part, the alignment of the text within the alphabetic entry field can be set to left-aligned, centered, or right-aligned. The default is left-aligned.

---

**Related Parts**

• **Currency Entry Field**
  Restricts input to currency related characters and displays input formatted as currency.

• **Date Entry Field**
  Restricts input to digits and displays input formatted as a date.

• **Entry Field**
  A general entry field part that allows information to be entered in any format.

• **Float Entry Field**
  Restricts input to floating point numbers.

• **Integer Entry Field**
  Restricts input to integer numbers.

• **Phone Number Entry Field**
  Restricts input to digits and displays input in phone number format.

• **Picture Entry Field**
  Restricts input to a pattern specified in the field's Properties dialog.
This example lets you enter information into all of the different kinds of formatted entry fields in the top of the window. The **changed:** event for each field is triggered when the field loses focus and the value has changed since the field gained focus. The event value (the unformatted contents of the field) is displayed in the entry field labeled changed. The changed field’s **setValue:** message fires a result link which sends the **valueFormatted:** message back to the formatted entry field. The **valueFormatted:** fires a result link that sends the **setValue:** message to the “valueFormatted:” entry field. The result of the **valueFormatted:** message (the formatted text) satisfies the argument of the **setValue:** message. This example is in the file FORMAT.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The Application part is a special part which serves as the foundation for everything you build with PARTS Workbench, including both finished applications and parts designed as components of other PARTS Workbench applications.

Unlike other parts, the application part does not have an icon in the Catalog which can be dragged and dropped. An icon isn't necessary because each workbench already includes one (and only one) application part which you can select by clicking on the workbench background—the empty space below the Tool bar and above the Information area. The application part fills this back-ground completely. When you drop a part onto the workbench, you are dropping it into an application part.

The application part is a nonvisual part in the sense that the user does not see it at runtime; the user sees visual parts such as windows and buttons that you place in the application part when you create the application. It does, however, affect the display of the visual parts it contains.

The origin of the application part (the upper-left corner of the workbench background) becomes the origin of the computer display when the application is executed. If you place a window very close to the origin of the application part when you build the application, the window is initially placed very close to the origin of the display when you execute the application.

Changing the size of the application part by resizing the workbench window can also affect the appearance of the windows and dialog windows. The effect on any individual window or dialog window depends on the Window layout settings for that part. You can view and change these settings via the Size window... button in its Properties dialog. The Window layout settings control whether or not the window’s position relative to the application part’s origin changes when you resize the application part, and whether or not the visual part’s size relative to the application part’s size changes when you resize the application part. See Appendix C, Standard Part Properties, for more information on Window layout settings.
Operation

You can save your application as a PAR file to be executed from the workbench, as a stand-alone executable (EXE) file and as an executable part (PAX). You can also use PAR files as nested parts in other PARTS Workbench applications by adding them to your Catalog or by importing them directly into your application. See the Nested Part and Part Accessor chapters later in this document for more details on reusing parts in other applications.

To create a new application:
> Double-click on the PARTS Workbench icon.

or

> Choose File/New from the workbench menu bar.

Either action opens a new workbench containing an empty application. You select the application by clicking on the workbench background between the Tool bar and Information area. Unlike other parts, you cannot delete an application part entirely. If you try to cut or delete an application part and confirm that you want to delete all parts, a new empty application part automatically replaces the old one.

You save an application in a part file so it can be used in other applications or edited again. If you plan to use the finished part in other applications you should read the Nested Part and Part Accessor chapters later in this document for more details.


See the file DELIVERY.TXT located in the DOC subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise installation directory for instructions on delivering an executable application.

Events

The application part has no events.

Messages

open

Opens the application. Has the same effect as pressing the Launch button or choosing Test/Launch from the workbench menu bar.

Returns: —

Triggers: —

Direct-Edit

There is no direct-edit for the application part.
Property Edit

Double-click on the workbench background to open the application part's Properties dialog. Each of the part-specific controls in the dialog is described in the following paragraphs.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The title of an application part becomes the default label for windows added to it.</td>
</tr>
<tr>
<td>Manufacturer</td>
<td>Enter text identifying the manufacturer of the part in this field. Its default value is taken from the User-name field in the PARTS Workbench Settings window. This information is a descriptive property available for other tool use. It is not used by PARTS Workbench itself.</td>
</tr>
<tr>
<td>Version</td>
<td>Enter text identifying the version of the part in this field. This information is a descriptive property, and is not used by PARTS Workbench itself.</td>
</tr>
<tr>
<td>Hint</td>
<td>Enter the hint for this application. The hint is shown in the Information area whenever the cursor is over the workbench background or the application icon in the Catalog.</td>
</tr>
</tbody>
</table>

Title

The title of an application part becomes the default label for windows added to it.

Manufacturer

Enter text identifying the manufacturer of the part in this field. Its default value is taken from the User-name field in the PARTS Workbench Settings window. This information is a descriptive property available for other tool use. It is not used by PARTS Workbench itself.

Version

Enter text identifying the version of the part in this field. This information is a descriptive property, and is not used by PARTS Workbench itself.

Hint

Enter the hint for this application. The hint is shown in the Information area whenever the cursor is over the workbench background or the application icon in the Catalog.
**Development Help file**

Enter the name of the help file, either a full pathname or a filename within the PARTS Workbench search path, containing development related help information for this application. By convention, it should contain an overview of the part plus its events and messages. This file is opened when the application is selected and the F1 key is pressed. If the part file is used as a nested part, this file is opened when the nested part is selected and the F1 key is pressed.

**Development Help topic**

Enter the help topic name in the help file for this application. It specifies the help panel to display when this application is selected and the F1 key is pressed.

**Part file reuse option**

This property specifies the intended use for this application if it is intended to be used as a part in another application.

- Packaged part (linked)—maintains a reference to the PAR file.
- Packaged part (embedded)—copies the application.
- Ensemble of parts—merges all the parts directly without packaging them.

The default is packaged part (linked). See the *Nested Part* chapter later in this document for more information on reuse options.

**Description...**

Opens a text pane that contains a textual description of the application. Any information that another user of this part might need to know can be entered here. If the application has previously been given a description, it can be read by opening this item. This information is a descriptive property, not used by PARTS Workbench itself.

**Related Parts**

- **Nested Part**
  You can reuse a part in another part by referencing it with nested part.

- **Part Accessor**
  Analogous to a runtime nested part. Used to improve application performance.

**Remarks**

There are some special considerations if you are delivering an application in PAR format which contains nested parts that specify the Packaged part (linked) reuse option. If you do not include the PAR files referenced by the nested parts, the application will not run. You must deliver the linked parts along with the application that contains them. These lower-level application files are not required if the applications are embedded or merged as an ensemble of parts.

When you save the application as an executable (EXE file) or as an executable part (PAX file), copies of all referenced part files are saved along with the parent part in a single file, so the individual files are no longer needed (the exception to this is parts
referenced by a part accessor which are loaded at runtime). Changes to referenced part files are not reflected when you run the EXE version, because it is constructed as a static snapshot.

The application part changes appearance when it is selected by displaying handles on the inside borders of the workbench workspace. The application is selected if no other parts are selected.

**Example**

This example shows an OK/Cancel dialog built as a separate part for use in other applications. It has an external `openTitled:` message to set the title of the dialog box to the argument with `setLabel:` and then `open` it. External `confirmed` and `cancelled` events are triggered when the matching button is pressed. Pressing either button also closes the dialog. See chapter 8, *Constructing New Parts*, in the *PARTS Workbench User's Guide* for information on creating new events and messages.

This example is in the file APPLICAT.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Array Holder

An array holder is a nonvisual part holding a fixed-sized collection of objects that stores and retrieves values using an integer index.

The array holder is the PARTS Workbench equivalent of an array variable in a traditional programming language. It is a value holder with the class permanently set to Array so that additional messages are available.

Operation

The array holder messages

- **at: anInteger**

and

- **at: anInteger put: aValue**

access and store the value corresponding to the index, *anInteger*. The value can be of any type. For each array holder element, you can also set an array holder to automatically generate events and messages that replace the **at: and at:put:** messages by checking the following property (a check box):

  Support element messages and events

Element messages look like this:

- **element**

and

- **setElement**:

The letter *n* in these messages is the integer index of the array holder element.

There are a number of other messages for manipulating and processing Arrays. Some of these are documented in the Messages section below.

Events

The array holder is a specialized value holder. In addition to the standard value holder events, it has the array holder specific events described in this section.

- **changedElement: aValue**

  This event is automatically generated for each element in the array holder if you choose the following array holder property: Support element messages and events (a check box). It occurs whenever the value corresponding to index *n* is changed.

  *Triggered By:*  
  **setElement**
  **at:put:**
Array Holder

indexException: anInteger
- Occurs when the array holder receives a message accessing the anInteger th element and anInteger is not a valid array holder element.
  Triggered By: at:put:

valueException: anArray
- Occurs when the array holder receives the replaceFrom:to:with: message and the replacement specified by with: is of the wrong size.
  Triggered By: replaceFrom:to:with:

Messages

The array holder is a specialized value holder. In addition to the standard value holder messages it has the array holder specific messages described in this section.

asArray
- Returns the contents of the array holder.
  Returns: an Array
  Triggers: —

asOrderedCollection
- Translates the contents of the array holder into a format suitable for an ordered collection part. You can place the result into an ordered collection holder by using a result link to the ordered collection holder's setValue: message.
  Returns: an Ordered Collection
  Triggers: —

asSet
- Translates the contents of the array holder into a member of class Set, eliminating any duplicate elements. You can create a part that supports the Set class by creating a value holder with its class set to Set.
  Returns: a Set
  Triggers: —

asSortedCollection
- Translates the contents of the array holder into a member of class SortedCollection containing the elements of the array holder sorted in ascending order. You can create a part that supports the SortedCollection class by creating a value holder with its class set to SortedCollection.
  Returns: a Sorted Collection
  Triggers: —

at: anInteger
- Returns the value stored at index anInteger. Triggers indexException: if the array holder index specified by anInteger does not exist.
  Returns: a Value
  Triggers: indexException:
at: anInteger put: aValue
Stores aValue at index anInteger. Triggers indexException: if the array holder index specified by anInteger does not exist.

Returns: —
Triggers: indexException:

concatenate: aCollection
Creates a new array containing the original followed by aCollection. Collections other than arrays (such as Dictionaries and Ordered Collections) are converted to Arrays before concatenation.

Returns: an Array
Triggers: —

copyFrom: beginning to: end
Creates a new array containing the elements of the receiver from index position beginning through index position end.

Returns: an Array
Triggers: —

copyReplaceFrom: beginning to: end with: aCollection
Creates a new array with the elements at index positions beginning through index position end replaced with the elements of aCollection. aCollection can be any type of Collection, not just an array.

Returns: an Array
Triggers: —

elementn
This message is automatically generated for each element in the array holder if you choose the following property: Support element messages and events (a check box). It returns the value at index n.

Returns: a Value
Triggers: —

first
Returns the first element of the array.

Returns: a Value
Triggers: —

includes: aValue
Returns true if the array holder contains an element equal to aValue, otherwise returns false.

Returns: a Boolean
Triggers: —

isEmpty
Returns true if the size of the array holder is zero, otherwise returns false.

Returns: a Boolean
Triggers: —
Array Holder

last

Returns the last element of the array.

Returns: a Value

Triggers: —

occurrencesOf: aValue

Returns the number of elements in the array holder that are equal to aValue.

Returns: an Integer

Triggers: —

reversed

Returns a new Array containing the elements of the original in reverse order.

Returns: an Array

Triggers: —

setElement n: aValue

This message is automatically generated for each element in the array holder if you choose the following property: Support element messages and events (a check box). It sets the value at index n to aValue.

Returns: aValue

Triggers: changedElement n:

size

Returns the number of elements in the array holder.

Returns: an Integer

Triggers: —

updateValue: anArray

Beginning with the first element in the array, updates each element in the array for which there is a corresponding element in anArray.

Returns: anArray

Triggers: valueException:

Direct-Edit

Direct-editing the array holder changes the part name that appears beneath the icon in the Workbench.

It is a good idea to change the part name to indicate the purpose of the array holder, just as you would use a descriptive name for a variable or a constant in a programming language.

Property Edit

The array holder’s Properties dialog is similar to the value holder’s with two exceptions: you can’t change the class of an array holder and the array holder has an additional option to generate messages and events for each element in the array holder.
The part-specific properties of the array holder are described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

### New value

This is an entry field that lets you type an expression for an array in the same format that would be used in a script. See chapter 9, *User-Defined Messages and Events*, in the *PARTS Workbench User's Guide*, for additional information.

### Edit value...

Press this button to display a dialog that lets you enter and edit values stored in the array holder. All values are stored in the array holder as Strings. If you want to store values other than Strings, use the New value field.

### Support element messages and events

If checked, one event, `changedElementn`, and two messages, `elementn` and `setElementn`, are automatically generated for each index position in the array holder. The messages can be used in place of the `at:` and `at:put:` messages. The default is to automatically generate element events and messages.
Related Parts

- **Value Holder**
  The array holder is similar to a value holder with its class set to Array. There are, however, some important differences: you cannot change the class of an array holder; a value holder cannot automatically generate element messages and events; and a value holder does not support Collection specific messages.

Remarks

The hint shown in the information area for an array holder shows the values the array holder currently contains.
**AVI Video**  
*(Win32 only)*

This nonvisual part is used to play an AVI (Audio / Video Interleaved) full-motion video file, which is displayed in a video pane.

To use this part, you also need a Media Control Interface (MCI) AVI video driver. This support is provided by Windows 95 and Windows NT; if you are running an older version of Windows, you must install the additional “Video for Windows” software. To hear the audio portion of the file, you must have a sound card and speakers.

**Operation**

The AVI video part is used to set the contents of a video pane. To make effective use of this part's capabilities, you will probably need to create an application with a visual interface, as shown in the Example topic later in this chapter.

The **open** message should be the first message sent to this part, and the **close** message should be the last message sent to it.

**Events**

An AVI video part has the following event.

- **playCompleted**
  - Indicates that playing (initiated by a **play**, **playFrom:** or **playFrom:**to:** message) has completed. Triggered by the MCI device driver when the end of the media is reached.
  - **Triggered By:** MCI device driver

**Messages**

An AVI video part has the following messages.

- **audio**
  - Answer whether audio output is enabled.
  - **Returns:** —
  - **Triggers:** —

- **audio:** aBoolean
  - Turn audio output on or off, according to aBoolean.
  - **Returns:** —
  - **Triggers:** —
close
Close the receiver.

Returns: —
Triggers: —

close

configure
Open the device configuration dialog.

Returns: —
Triggers: —

filename
Answer the current filename of the .AVI file.

Returns: a Filename
Triggers: —

filename: aString
Set the current filename to aString.

Returns: —
Triggers: —

isNotReady
Answer whether the device state is 'Not Ready'.

Returns: a Boolean
Triggers: —

isOpen
Answer whether the receiver is open.

Returns: a Boolean
Triggers: —

isPaused
Answer whether the device state is 'Paused'.

Returns: a Boolean
Triggers: —

isPlaying
Answer whether the device state is 'Playing'.

Returns: a Boolean
Triggers: —

isReady
Answer whether the device state is 'Ready'.

Returns: a Boolean
Triggers: —

isSeeking
Answer whether the device state is 'Seeking'.

Returns: a Boolean
Triggers: —
isStopped
   Answer whether the device state is 'Stopped'.
   Returns: a Boolean
   Triggers: —

isTimeFormatFrames
   Answer whether the time format is Frames.
   Returns: a Boolean
   Triggers: —

isTimeFormatMilliseconds
   Answer whether the time format is Milliseconds.
   Returns: a Boolean
   Triggers: —

length
   Answer the length of the receiver, in the current time format.
   Returns: a String
   Triggers: —

open
   Open the receiver, and display the first frame.
   Returns: —
   Triggers: —

pause
   Pause the receiver.
   Returns: —
   Triggers: —

play
   Begin playing the receiver starting at the current position.
   Returns: —
   Triggers: —

playFrom: startFrame
   Begin playing the receiver starting at the position indicated by startFrame.
   Returns: —
   Triggers: —

playFrom: startFrame to: endFrame
   Begin playing the receiver starting at the position indicated by startFrame, and stop playing when the position reaches endFrame.
   Returns: —
   Triggers: —

position: anIntegerOrString
   Set the current position to anIntegerOrString (in units of the current time format). A String is permitted so that non-integer time formats can be specified.
   Returns: —
   Triggers: —
reset
Reset the position to the beginning.

Returns: —
Triggers: —

resume
Resume playback from the current position, following a pause.

Returns: —
Triggers: —

reverse
Answer whether playback is to be reversed.

Returns: a Boolean
Triggers: —

reverse: aBoolean
Set whether playback is to be reversed, according to aBoolean.

Returns: —
Triggers: —

speed
Answer the playback speed ratio (where 1 is normal, less than 1 is slower, and greater than 1 is faster).

Returns: a Number
Triggers: —

speed: aNumber
Set the playback speed ratio to aNumber (where 1 is normal, less than 1 is slower, and greater than 1 is faster).

Returns: —
Triggers: —

stop
Stop playing.

Returns: —
Triggers: —

timeFormat
Answer a String which indicates the current time format.

Returns: a String
Triggers: —

timeFormat: aTimeFormatString
Set the current time format to aTimeFormatString.

Returns: —
Triggers: —
timeFormatFrames
Set the time format to Frames.

Returns: —
Triggers: —
**timeFormatMilliseconds**
Set the time format to Milliseconds.

*Returns:*  
*Triggers:*  

**video**
Answer whether video output is enabled.

*Returns:* a Boolean  
*Triggers:*  

**video: aBoolean**
Turn video output on or off, according to aBoolean.

*Returns:*  
*Triggers:*  

**Direct-Edit**
Direct-editing the AVI video part lets you change the part name (directly on the workbench).

**Property Edit**
The part-specific property of the AVI video part is described in the following paragraph. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties.*

**File Name...**
This push button opens a standard file browser so you can locate and select the filename of the AVI (Audio / Video Interleaved) full-motion video file.

**Related Parts**
- **Video Pane (Win32 only)**
The video pane is a visual part that is used to display AVI video files.
Example

This example demonstrates a visual interface to the AVI Video part's capabilities. The File button opens a dialog where you select the desired file. Other push buttons control the playing of the file, and information about the selected file is displayed. (You could provide a fancier interface by replacing the labeled push buttons with a toolbar having VCR-style bitmap graphics.)

Check boxes determine whether audio and video output are enabled, and whether the video is stretched to fit the video pane. When the Stretch feature is checked, scroll bars in the video pane allow the video to be positioned within the frame of the pane.

This example is in the file VIDEOX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)

For further details about the video pane part used in this example, see the example in the Video Pane (Win32 only) chapter later in this document.
**Btrieve Accessor**

The Btrieve accessor is a nonvisual part that lets you create a Btrieve database or access an existing Btrieve database.

**NOTE:** In the initial release of version 3.1.0 of Visual Smalltalk or Visual Smalltalk Enterprise, the Btrieve accessor part does not run on Windows NT or Windows 95.

Btrieve is a key-based database. One or more of the database record fields are designated as keys and every database access operation uses a key. The Btrieve accessor can automatically generate visual parts to match each of the database's fields.

Btrieve records are sorted in key order. When there are duplicate keys, the tie is broken by insertion order. This ordering scheme is used by all the `get` messages when searching.

**Operation**

The database layout is defined through the Btrieve accessor Properties dialog. There are two ways to define a record: select a visual part to define each field, or define the field properties for each field first and then automatically generate the required visual parts. In either case, the links between the visual parts and the events and messages for the corresponding record fields are automatically generated. These events and messages allow you to display and update record fields with visual parts.

The Btrieve accessor automatically generates an event for every field in the record definition. This event is triggered whenever the record buffer is updated as a result of a change in the current database position. The event value is the field's contents for the record at the current position.

Messages are provided to search the database by key value, traverse it sequentially, and insert, delete, and update records.

Two additional messages are automatically created for every field. One of the messages, `setfield`, sets the contents of the corresponding field in the record buffer to the argument of the message. The other message has the same name as the field and returns the value of the field in the current record buffer. An additional message, `usefieldAsKey`, is automatically created for each key field in the database record definition. The first key field created is used as the key until the key field is changed with the `usefieldAsKey` message. In the above examples, the actual field name is substituted for `field` in the message name.
The argument type for the field messages depends on the type of the corresponding part. The table below shows the correspondence between parts, database record field types, and PARTS Workbench data types. Only the visual parts listed in this table generate database fields.

<table>
<thead>
<tr>
<th>Visual Part</th>
<th>Btrieve Field Type</th>
<th>Record Field Length in Bytes</th>
<th>Corresponding PARTS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabetic entry field</td>
<td>Fixed-Length String</td>
<td>64 ¹</td>
<td>String</td>
</tr>
<tr>
<td>check box</td>
<td>Two-Byte Boolean</td>
<td>2</td>
<td>Boolean</td>
</tr>
<tr>
<td>combo box</td>
<td>Fixed-Length String</td>
<td>64 ¹</td>
<td>String</td>
</tr>
<tr>
<td>currency entry field</td>
<td>Double-Precision IEEE float</td>
<td>8</td>
<td>Fraction</td>
</tr>
<tr>
<td>date entry field</td>
<td>Date</td>
<td>4</td>
<td>Date</td>
</tr>
<tr>
<td>dial pane</td>
<td>Four-Byte Signed Integer</td>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>drop-down list</td>
<td>Four-Byte Signed Integer</td>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>entry field</td>
<td>Fixed-Length String</td>
<td>64 ¹</td>
<td>String</td>
</tr>
<tr>
<td>float entry field</td>
<td>Double-Precision IEEE float</td>
<td>8</td>
<td>Float</td>
</tr>
<tr>
<td>graph pane</td>
<td>Any Object</td>
<td>variable ²</td>
<td>Bitmap</td>
</tr>
<tr>
<td>integer entry field</td>
<td>Four-Byte Signed Integer</td>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>list pane</td>
<td>Four-Byte Signed Integer</td>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>phone number entry field</td>
<td>Fixed-Length String</td>
<td>64 ¹</td>
<td>String</td>
</tr>
<tr>
<td>picture entry field</td>
<td>Fixed-Length String</td>
<td>64 ¹</td>
<td>String</td>
</tr>
<tr>
<td>text pane</td>
<td>Variable-Length ZString</td>
<td>variable ²</td>
<td>String</td>
</tr>
</tbody>
</table>

Note:
1) The initial size given to the Fixed-Length String fields can be modified by selecting the field in the Btrieve accessor and opening the Field Properties dialog.
2) Up to 64,000 bytes in size for OS/2; up to 32K bytes in size for Windows.
   Note that this size limit is also the limit for all fields combined.

A default file can be associated with the database, or a filename can be provided when the database is opened.

**NOTE:** Once a database file has been created, the record layout for that file can’t be changed.

A database file is automatically created if the Btrieve accessor is opened on an empty or non-existent file.
Events

**endOfFileException**

Occurs when one of the `get` messages listed below fails because no corresponding record was found. If this event is not linked, a warning dialog is displayed when an end of file exception occurs. If you do not want this dialog to appear, you must create a link from this event.

**Triggered By:**
- `getFirst`
- `getGreaterOrEqual`
- `getLast`
- `getLessOrEqual`
- `getNext`
- `getPrevious`

**exception:** *anInteger*

Occurs when a database operation fails. If an operation fails because the end of file is reached, the `endOfFileException` event is triggered instead. The event value is the Btrieve status code. This code can be converted to text with the `exceptionString:` message. If this event is not linked, a warning dialog is displayed when an exception occurs. If you do not want this dialog to appear, you must create a link from this event.

**Triggered By:**
- any message

**field:** *aValue*

Occurs when the record buffer changes because of a database operation and a new value is available for the named field.

**Triggered By:**
- `delete`
- `getEqual`
- `getFirst`
- `getGreaterOrEqual`
- `getLast`
- `getLessOrEqual`
- `getNext`
- `getPrevious`

Messages

**close**

Closes the database file.

**Returns:** —

**Triggers:**`exception:`

**delete**

Deletes the current record and clears all the fields in the record buffer.

**Returns:** —

**Triggers:**
- **field:**
  - `exception:`
exceptionString: anInteger

Returns a textual explanation of an error code provided by the exception: event. See the Remarks section later in this chapter for a list of error codes and messages.

Returns: a String
Triggers: —

triggers:

field

Automatically generated for each field in the database, where field is the name of a database field. Returns the value of the named field in record buffer.

Returns: a Value
Triggers: exception:

getEqual

Finds the first record whose key matches the current key field. Ordering is based on the order of insertion into the database. Triggers exception: with the Key value not found message if there are no matching records in the database.

Returns: —
Triggers: field:

getFirst

Uses the current key field to find the record with the lowest key value. Triggers endOfFileException if there are no records in the database.

Returns: —
Triggers: field:
endOfFileException

getGreaterOrEqual

Finds the first record whose key is not less than the value in the key field of the buffer. Ordering is based on the order of insertion into the database.

Returns: —
Triggers: field:
endOfFileException

getLast

Uses the current key field to find the record with the highest key value.

Returns: —
Triggers: field:
endOfFileException

getLessOrEqual

Finds the first record whose key is not greater than the value in the key field of the buffer. Ordering is based on the order of insertion into the database.

Returns: —
Triggers: field:
endOfFileException
getNext
Advances to the next record.

Returns:
—
Triggers:

field:
endOfFileException
exception:

getPrevious
Advances to the previous record that matches the current key field.

Returns:
—
Triggers:

field:
endOfFileException
exception:

insert
Creates a new record from the buffer's contents and inserts it following the current record.

Returns:
—
Triggers:

exception:

lock
Locks the current record for updating.

Returns:
—
Triggers:

exception:

open
Opens the default database file for access. Creates a new file with the default file name if one does not already exist.

Returns:
—
Triggers:

exception:

openOn: aString
Opens the database file named aString.

Returns:
—
Triggers:

exception:

set Field: aValue
Automatically generated for each field in the database, where Field is a database field name. Sets the value of the named field in the record buffer to the argument.

Returns:
—
Triggers:

exception:

unlock
Unlocks the currently locked record.

Returns:
—
Triggers:

exception:

update
Writes the current record buffer to the database.

Returns:
—
Triggers:

exception:
**useFieldAsKey**

Sets the field to be searched by `get` messages.

- **Returns:** —
- **Triggers:** `exception`:

**write**

Inserts a record if it is new or updates the record if it already exists. A record already exists if inserting it would cause a unique key field to have duplicate values. If the record already exists, Btrieve accessor will use the current key field to find the record which will be updated.

- **Returns:** —
- **Triggers:** `exception`:

### Direct-Edit

Direct-editing the Btrieve accessor changes the part name that appears beneath the icon in the workbench.

### Property Edit

Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

#### Default file name

Enter the name of the file to be used when the database is opened with the `open` message.
Fields

The fields in the record definition appear in the list pane. New fields can be added or existing fields can be removed or revised. Once a database file has been created, its record definition cannot be changed.

To add a new field to the record definition:

1. Click on the Create new field... button.
   The Create Field dialog opens.

2. Enter the field name.
   This name becomes the name of a new event and message added to the Btrieve accessor to access this field.

3. Select the type of the field from the Field type list.

4. If the size is not determined by the type, enter the size of the field.

5. If the field is a key, click the Is key check box.

6. If the field is a key and only one record is permitted with that key value, check the Unique key values check box.

7. Click the OK button.

Repeat the previous steps to define more fields.

To define fields based on visual parts in your application:

1. Drag an appropriate part for each field into your application.

2. Give each part a name that reflects its use in the database.
   These names are used for the field names when the Btrieve accessor is created.

3. Drag the Btrieve accessor icon to your application.

4. Open the Btrieve accessor’s Properties dialog.
5 Click on *Pick fields* with mouse. The cursor changes to a pointing-hand icon and the Workbench is brought to the front.

6 Click on each of the visual parts that represents a database field. Use the angle bracket at the end of the finger shaped cursor to designate each part.

7 Click outside the workbench or press the Escape key when you are finished. The database fields will be named with the part names of the corresponding visual parts, and the type of the database fields will be based on the kind of visual part chosen (see the table in the Operations section earlier in this chapter).

An alternate way to create a record definition from visual parts is to choose the part names from a selection list:

1 As above, add the appropriate parts to your application, name them, and open the Btrieve Properties dialog.

2 Click on the *Pick fields from list...* button. A multiple selection list appears (as shown in the following illustration) with the names of all visual parts that can be used for database fields selected.

3 Deselect those parts that you do not want to include in the record definition and press OK.

To delete a field from the record definition:

1 Select the field from the list of fields.

2 Press the Remove field button.
To modify a field definition:
1. Select the field from the list of fields.
2. Press the Properties... button.
3. Revise the field properties shown in the dialog (see Setting Field Properties later in this chapter).

You cannot change the field name.

Create Form Parts...
You can use the record definition to automatically create visual parts that allow you to enter and display the information in each database field.

When you press the Create form parts... button, you are presented with a list of eligible parts in your application.

Use the Exclude field button to remove parts that you don't want to use to make database fields. Arrange the remaining parts by dragging each field higher or lower in the list. List position determines how the resulting parts will be positioned in the workbench. Click on OK when finished.

Click on the pane that is to be the parent of these parts. A new visual part is created for each field selected. Links are automatically created between the parts and the Btrieve accessor and the parts are labeled with static text containing the field name.

Setting Field Properties
The dialog shown below is used to define or revise the properties for a field.

Field name
Enter the field name. This can only be done when creating a field; the name cannot change when editing properties of an existing field.
Field type

Choose the Btrieve database type for the field from this list. The following table shows the attributes of each field type as well as the data type of the argument expected and returned by the automatically generated events and messages corresponding to that field.

Table of Field Types, Sizes, and Corresponding PARTS Types.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Length in Bytes</th>
<th>Corresponding PARTS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any Object</td>
<td>variable (up to 64KB)</td>
<td>Any Value</td>
</tr>
<tr>
<td>Date</td>
<td>4</td>
<td>Date</td>
</tr>
<tr>
<td>Double-Precision IEEE Float</td>
<td>8</td>
<td>Float</td>
</tr>
<tr>
<td>Fixed-Length Byte Array</td>
<td>user specified</td>
<td>String</td>
</tr>
<tr>
<td>Fixed-Length String</td>
<td>user specified</td>
<td>String</td>
</tr>
<tr>
<td>Four-Byte Signed Integer</td>
<td>4</td>
<td>Integer</td>
</tr>
<tr>
<td>Time</td>
<td>4</td>
<td>Time</td>
</tr>
<tr>
<td>Two-Byte Boolean</td>
<td>2</td>
<td>Boolean</td>
</tr>
<tr>
<td>Two-Byte Signed Integer</td>
<td>2</td>
<td>Integer</td>
</tr>
<tr>
<td>Variable-Length ZString</td>
<td>variable (up to 64KB)</td>
<td>String</td>
</tr>
</tbody>
</table>

Size in bytes

Enter the size for fields of type Fixed-Length Byte Array and Fixed-Length String.

Is key

Check this box if the field is to be used as a key for a database search.

Unique key values

Check this box if this field is a key and only one record in the database can have any particular value.

Remarks

You must use the Btrieve installation procedure listed in Getting Started before using the Btrieve accessor.

The Btrieve accessor uses licensed programs. Before distributing an application containing a Btrieve accessor, read the file DELIVERY.TXT in your PARTS Workbench directory for additional information.

The database file is created when the database is opened if it didn't previously exist.

When a field is defined by a visual part, the defining part is remembered and recreated if the Create form parts... button is chosen and that field is selected.
All variable-length fields are placed at the end of the record definition. The Properties dialog displays the field names in the order they appear in the record definition.

The Btrieve accessor triggers the exception event when an exceptional condition occurs. The error code provided can be translated into a string using the exceptionString message. The messages that can occur are listed in the following paragraphs, ordered by the error code number.

**File not open**
Occurs when the database is accessed without first being opened.

*Error Code:* 3

**Key value not found**
Occurs when getEqual is executed and no matching record is found.

*Error Code:* 4

**Duplicate key value**
Occurs when you attempt to insert or update a record that has a value in the unique key field that already exists in the database.

*Error Code:* 5

**Key undefined**
Occurs when you attempt to access a record by key but no key field was defined.

*Error Code:* 6

**After changing the key, before sending either getNext or getPrevious, you must use another type of a get operation**
You should establish a new starting position after changing the key field.

*Error Code:* 7

**No current position has been established**
The database must be positioned to a record before a delete or update operation is performed.

*Error Code:* 8

**No more records**
Occurs when getNext or getPrevious is executed and no more records match the key value.

*Error Code:* 9

**Filename does not conform to file naming conventions**
Occurs when the database is opened and either the default filename or the argument of openOn is in the legal format for your file system.

*Error Code:* 11

**Filename specified does not exist**
Occurs when the database file can't be found because of incorrect drives or directories in the path name.

*Error Code:* 12
Disk is full
Occurred when there is no more room on the device containing the database file.

Error Code: 18

Unrecoverable error
Occurred when a system error occurs while accessing the database file.

Error Code: 19

Data Buffer Length
Can occur when you open a new file without any fields defined.

Error Code: 22

Invalid page size, the following line must be in the WIN.INI:
OPTIONS=M: 64/P: 4096
or
Invalid page size, the following line must be in the CONFIG.SYS: BTRPARMS=M: 64/P: 4096
For Windows, your WIN.INI file must contain this OPTIONS line in order to use the Btrieve accessor.
For OS/2, this BTRPARMS line must appear in the CONFIG.SYS file.

Error Code: 24

Unable to create file
A directory in the path specified does not exist or the filename is not valid.

Error Code: 25

Not a Btrieve file
Occurred when the Btrieve accessor is opened on an existing file that isn't in the proper Btrieve format.

Error Code: 30

Record locked
Occurred when you try to update a record that is locked.

Error Code: 84

Permission error
Your application tried to open or create a file in a directory without the proper privileges.

Error Code: 94

The memory parameter of OPTIONS environment variable is too small
or
The memory parameter of BTRPARMS environment variable is too small
For Windows, increase the size of the memory parameter on the OPTIONS statement in your WIN.INI file. It should be set to at least /M:64.
For OS/2, increase the size of the memory parameter on the BTRPARMS statement in your CONFIG.SYS file. It should be set to at least /M:64.

Error Code: 1003
Incorrect page size parameter
For Windows, set the page size parameter of the OPTIONS statement in your WIN.INI file to /P:4096.

For OS/2, set the page size parameter of the BTRPARMS statement in your CONFIG.SYS file to /P:4096.

Error Code: 1004

Unknown parameter in the OPTIONS environment variable
or

Unknown parameter in the BTRPARMS environment variable
For Windows, check the OPTIONS statement in your WIN.INI file for incorrect parameters.

For OS/2, check the BTRPARMS statement in your CONFIG.SYS file for incorrect parameters.

Error Code: 1008

Transaction control files not supported. Remove the /T: parameter from the [btrieve]
section of your WIN.INI file
or

Transaction control files not supported. Remove the /T: parameter from the BTRPARMS
in your CONFIG.SYS file
Transaction control files are not supported.

For Windows, remove the /T: <file> specification in the [btrieve] section of your WIN.INI file.

For OS/2, remove the /T: <file> specification in the BTRPARMS statement in your CONFIG.SYS file.

Error Code: 1009

Object too large for field. Object will be truncated
Occurs when the argument of the Btrieve accessor part’s #setField: message is bigger than the field size specification.

Error Code: 9997

Unknown error occurred while requesting a field value
Triggered when the Btrieve accessor is unable to determine the exact nature of the error that has occurred.

Error Code: 9998

Type Mismatch
The data does not match the field type (e.g. attempting to put a value that doesn’t conform to the date format into a date field).

Error Code: 9999
Example

This example helps you familiarize yourself with the operation of a Btrieve database. The simple database has two fields: Name and Address. The name field is the default key, but you can switch the key field to the address field by pressing the useAddressAsKey button.

Before running this example, you should edit the properties of the Btrieve accessor and set the database filename to something appropriate for your system. See Getting Started for instructions on setting up Btrieve before you run this application.

If an exception occurs, a default information dialog appears to explain the problem.

This example is in the file BTREXmpl.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A button list box is a visual part that provides a general selection capability similar to a multiple choice list box, but each item in the list includes a check box (with either two or three states) in addition to the text string.

This part implements a control described in the user interfaces guidelines for Windows 95. The Windows version of this part supports an optional image (16 x 16 small icon or bitmap) for each item in the list, as shown in the following illustration:

### Operation

The button list box allows the user to select multiple items from the list by clicking the check box for each desired item. This provides an alternative to Shift-clicking and Ctrl-clicking.

If the Three State property is checked, the check boxes have an additional state of indeterminate. (Even if this property is not checked, an item's state can be set programmatically to the indeterminate state using the `indeterminateIndex:` or `indeterminateItem:` messages.)

You can also use the scroll bar or arrow keys to scroll through the items, and press the space bar to select an item when it is highlighted.

### Events

A button list box has the following events.

**aboutToChange**

Occurs when a change of selection is requested. If the `abortChange` message is executed during the firing of any link connected to this event, the selection is canceled and the `changed:` event is not triggered.

*Triggered By:* the user

*linkValue:*

*setValue:*

*setValueIndex:*
changed: anObject
Occurs whenever the selection changes. The event value provides a list of
the currently selected items.
Triggered By: the user
setValue:
setValueIndex:

changedIndex: anInteger
Occurs when the selection changes. The event value provides the index of
the currently selected item.
Triggered By: the user
setValue:
setValueIndex:

checked: anObject
The list item specified by the event argument was checked by the user.
Triggered By: the user

checkedIndex: anInteger
The list index specified by the event argument was checked by the user.
Triggered By: the user

doubleClicked: anObject
Occurs when the user double-clicks on an item. This action triggers the
changed: and changedIndex: events first, then the doubleClicked: event.
The event value is the newly selected item.
Triggered By: the user

indeterminate: anObject
The list item specified by the event argument was changed to the
indeterminate state.
Triggered By: indeterminateItem:

indeterminateIndex: anInteger
The list index specified by the event argument was changed to the
indeterminate state.
Triggered By: indeterminateIndex:

needsContents
Occurs when the button list box is about to be opened, to request that its
contents be set. An event handler would respond with the #contents:
message.

needsImageFor: anObject
The control needs the optional image for anObject. The event handler should
return a 16 x 16 bitmap or small icon.

needsStateFor: anObject
The control needs the state for anObject. The event handler should return a
value of true (for checked), false (for unchecked), or nil (for indeterminate)
to indicate the item’s initial state. If not handled, the item’s state is assumed
to be unchecked.
needsStringFor: anObject
The control needs the displayable string for anObject. The event handler
should return a String. If not handled, the #asString message is used.

rightClicked
Occurs when the user clicks on the button list box with mouse button 2.
Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the control has the input focus. You
can connect this event to the setFocus message of another part to set the tab
order.
Triggered By: the user

unchecked: anObject
The list item specified by the event argument was unchecked by the user.
Triggered By: the user

uncheckedIndex: anInteger
The list index specified by the event argument was unchecked by the user.
Triggered By: the user

Messages
A button list box has the following messages.

abortChange
Cancels a selection if executed as a consequence of triggering the
aboutToChange event.
Returns: —
Triggers: —

checkIndex: anInteger
Set the checked state for the item with the given index.
Returns: —
Triggers: —

checkItem: anObject
Set the checked state for the given item.
Returns: —
Triggers: —

deleteAll
Removes all items from the selection list.
Returns: —
Triggers: —

deleteItem: anObject
Removes the specified item from the selection list.
Returns: —
Triggers: —
Button List Box

**disable**
Ignores user selections.

Returns:  
Triggers: 

**enable**
Enables user selection.

Returns:  
Triggers: 

**indeterminateIndex:** **anInteger**
Set the indeterminate state for the item with the given index.

Returns:  
Triggers: 

**indeterminateItem:** **anObject**
Set the indeterminate state for the given item.

Returns:  
Triggers: 

**insertItem:** **anObject**
Appends **anObject** to the selection list.

Returns:  
Triggers: 

**list**
Returns the items in the selection list.

Returns:  an Array of Objects
Triggers: 

**setFocus**
Moves the input focus to the button list box.

Returns:  
Triggers: 

**setList:** **anArrayOfObjects**
Sets the items in the selection list to the items in the argument and returns the new list.

Returns:  
Triggers: 

**setValue:** **anObject**
Sets the selection to **anObject**.

Returns:  
Triggers:  

**setValuelIndex:** **anInteger**
Sets the selection to the item with the index **anInteger**.

Returns:  
Triggers:  

Visual Smalltalk Enterprise Part Reference
**uncheckIndex:** *anInteger*
Set the unchecked state for the item with the given index.

*Returns:* —
*Triggers:* —

**uncheckItem:** *anObject*
Set the unchecked state for the given item.

*Returns:* —
*Triggers:* —

**value**
Returns the currently selected item.

*Returns:* an Object
*Triggers:* —

**valuelIndex**
Returns the index of the currently selected item.

*Returns:* an Integer
*Triggers:* —

**Direct-Edit**
Direct-editing the button list box lets you enter the list of items. Use the Enter key to separate items. Click outside the button list box when you are done entering items.

**Property Edit**
Each of the part-specific controls in the button list box's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties.*
Three State
When checked, this property allows the check box for each item in the list to have a third state of indeterminate. By default, the check boxes have the two states of checked or unchecked.

Use Images
The Windows version of this part supports an optional image (16 x 16 small icon) for each item in the list. When this property is enabled (which it is by default), the image for each item in the list is displayed between the check box and the text string.

The image is supplied as a Bitmap or an Icon in response to the needsImageFor: or needsSelectedImageFor: messages.

Related Parts
• List Pane
  The list pane is a visual part that allows you to make a single selection from a scrollable list of items.

• Multiple Choice List
  The multiple choice list is a visual part that provides a general selection capability, allowing more than one item to be selected at a time.
Example

This example demonstrates the use of the Button List Box part.

This example is in the file BTNLISTX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)
The C structure, a nonvisual part, makes procedure calls to C dynamic link libraries which require arguments of C structure (reference) type.

The fields of a C structure correspond to the fields in a DLL. You declare the fields of the C structure by using the Properties dialog. Once fields are declared, the C structure part automatically creates events and messages to access and set the field contents.

For convenience in describing these automatically generated events and messages in the rest of this reference, the names `field` and `field:` are used as a shorthand notation. The name of each structure field should be substituted for `field`.

**Operation**

When using the C structure, you have to be careful that you align the fields in your C structure the same way they are aligned in the DLL. C structure does not automatically byte-align or word-align fields, because it is impossible to tell ahead of time which compiler and options you are using, and therefore there is no way of knowing how your DLL is aligned. If the fields in your C structure are not aligned the same way they are aligned in the DLL, errors can occur.

Recompiling a 16-bit DLL using a 32-bit compiler is an example of a situation that can cause field alignment problems. For example, say you have a C header file which looks like this:

```c
typedef struct _cstruct {
    SHORT shortInt;
    CHAR *stringPointer;
    CHAR arrayOfChar[10];
    ULONG uLongInt;
} cstruct, *pcstruct;
```

Your corresponding C structure part would be defined this way:

<table>
<thead>
<tr>
<th>Field name</th>
<th>Field type</th>
<th>Field length</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortInt</td>
<td>unsigned short</td>
<td>2</td>
</tr>
<tr>
<td>stringPointer</td>
<td>reference</td>
<td>4</td>
</tr>
<tr>
<td>arrayOfChar</td>
<td>array of characters</td>
<td>10</td>
</tr>
<tr>
<td>uLongInt</td>
<td>unsigned long</td>
<td>4</td>
</tr>
</tbody>
</table>
When a C program using pcstruct is compiled into a DLL using a 16-bit compiler, the C structure works correctly. This is because all the fields in the C structure contain an even number of bytes, and the 16-bit compiler is aligning the fields along 2-byte boundaries. Therefore, the field offsets within the C structure are the same as the field offsets within the DLL.

However, if you now compile the C file with a 32-bit compiler, and you take no extra steps to pack the data structure (more on this below), the compiler will align these fields along 4-byte boundaries, padding the structure with extra bytes. Because C structure does not automatically align the fields, they won’t align properly:

<table>
<thead>
<tr>
<th>Id</th>
<th>Offset in CStruct</th>
<th>Offset in DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortInt</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>stringPointer</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>arrayOfChar</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>uLongInt</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

In this example, accessing any field except the first field could cause a crash because the offset within the C structure does not match the actual data offset when compiled.

If you have a situation where the offsets within your C structure do not match the offsets within your DLL, you can address it in one of two ways:

1. Recompile the DLL

   If you are using a DLL which you compiled yourself, you can address this situation by compiling the DLL differently. Most compilers provide a way of packing your structures when compiled, which causes the structures to be compiled without any padding. This is usually either a runtime option to the compiler or some compiler directive in your .H file. For example, using a 32-bit compiler that understands the #pragma pack directive, the resulting DLL will contain packed data structures if the structure is defined this way:

   ```c
   #pragma pack(1) /* force structure alignment packing along 1-byte boundaries*/
   typedef struct _cstruct {
     SHORT shortInt;
     CHAR *stringPointer;
     CHAR arrayOfChar[10];
     ULONG uLongInt;
   } cstruct, *pcstruct;
   #pragma pack() /* restore packing to default */
   ```

   Note that the customary default is unpacked, so if you need to pack your data structures you must consult the documentation provided with your compiler for how to pack your data structures.
2. Pad your C Structure

If you cannot or do not wish to recompile your DLL, you can address this situation in PARTS Workbench. Create your C structure data fields such that they are aligned in the same way your compiler aligns the corresponding fields. In some cases, simply rearranging the fields could accomplish this. In other cases, it will be necessary to provide padding fields yourself. For instance, in the previous example, you could define the C structure part as follows:

```
shortInt  unsigned short  2
padding1   array of characters  2
stringPointer reference  4
arrayOfChar array of characters  10
padding2   array of characters  2
uLongInt   unsigned long  4
```

Now your fields will line up like this:

```
Field  Offset in CStruct  Offset in DLL
shortInt    0              0
padding1    2              4
stringPointer 4              4
arrayOfChar  8              8
padding2  18
uLongInt  20              20
```

**Events**

One event is automatically generated for each field. The event is named the same as the field, and ends with a colon because the field value is supplied as the event value.

**field: aValue**

Occurs whenever a DLL accessor message returns that was passed a reference to this C structure as an argument. You should link to this event for any fields that are updated by the DLL procedure to get the new values.

Triggered By:  external DLL return
Messages

Two messages are automatically generated for each C structure field, one to set the field value and one to get the field value. Both messages are named the same as the field. The setting message ends with a colon because it requires an argument.

field

Returns the current value of the specified field.

- Returns: a Value
- Triggers: —

field: aValue

Sets the specified field to aValue.

- Returns: aValue
- Triggers: —

free

When a call is made to a DLL accessor using a C structure, global memory is allocated for the C structure. When the user is finished with this memory, he should send the free message to the C structure to free the memory. Otherwise, the memory is not freed until the image is shut down.

If the user runs multiple applications in an image, any application that includes a C structure will remain in the image until its memory is freed. This includes any portion of the application using part accessors; even if it is unloaded, the part accessor's portion of the application remains in memory until the C structure is freed. The aboutToUnload event of the part accessor provides an opportunity for the application to free its C structures.

- Returns: —
- Triggers: —

setFromReference: aReference

When you have a DLL accessor message which returns a pointer to a C struct, use this message to get the contents of that C struct. Link the result of the DLL accessor message to the C Structure's setFromReference: message. The contents of the C Structure part will be updated using the result of the DLL call, and all the field events will be triggered.

- Returns: —
- Triggers: all the field events

directFromReference

value

Returns a reference to the C structure. This reference value is used as an argument to any DLL accessor message that invokes a DLL procedure that requires an argument of reference type.

- Returns: a reference
- Triggers: —

Direct-Edit

Direct-editing the C structure changes the part name that appears beneath the icon in the workbench.
Property Edit

The Properties dialog for the C structure part is shown below. Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Fields

This list pane shows the fields that have been defined and their order.

Create field...

Opens the Create Field dialog that lets you define a new field in the structure. This dialog is also used when you choose Properties... to modify the properties of fields that have already been defined.

Field name

Enter the name of the new field.
**Field type**
Select the C data type for the field. The possible choices are:

- **short** A 16-bit signed integer. Use Integer arguments for the matching event and messages.
- **ushort** A 16-bit unsigned integer. Use Integer arguments for the matching event and messages.
- **long** A 32-bit signed integer. Use Integer arguments for the matching event and messages.
- **ulong** A 32-bit unsigned integer. Use Integer arguments for the matching event and messages.
- **double float** An 8 byte value. A double float supports IEEE double float. Use a float argument.
- **array of char** A fixed length array of characters. Use String arguments for the matching event and messages. You must enter the size of the array in the Size in bytes field.
- **reference** A reference field is useful when you have a C struct field containing a pointer to another C struct or a pointer to an array of characters. A reference field uses 4 bytes.
  - If this field is a pointer to a second C struct, set this field to the result of the value message to the second C Structure part.
  - If this field is a pointer to an array of characters, set this field to the result of the value message of a string holder part. Make sure the string holder contains a string of appropriate length before value message executes.

**Size in bytes**
This field shows the size of the selected field type. If you choose array of char for the field type, you must enter the length of the array in this field before pressing OK.

**Remove Field**
Removes the field selected in the Fields pane in the Properties dialog.

**Properties...**
Opens the Field Properties dialog for the currently selected field, allowing you to change the properties described under Create field... above.

**Create form parts...**
You can create visual parts that correspond to the fields in the C structure. The events and messages for the fields are automatically connected to the value and setValue: messages of the new visual parts.

When you press the Create form parts... button, a Form Item Layout dialog is displayed:
The list contains the fields of the C structure. Select any fields for which you don’t want to generate visual parts and click on the Exclude field button. When you are done excluding fields, click on the OK button. The Workbench window will come to the front and the cursor will look like a part icon. Click on the pane that you want to contain the generated parts. The visual parts will be positioned in the pane and links will be automatically created between the C structure fields and the corresponding visual parts.

**Related Parts**

- **DLL Accessor**
  The C structure must be used in combination with the DLL accessor. In particular, it is used to access the fields of arguments declared to be of reference type in the DLL accessor.

**Remarks**

In order to rename a field, you must remove it and recreate it. You cannot create a field in the middle of a C structure; every new field that you create is appended to the end of the C structure.
This nonvisual part is used to play CD audio discs, using the Media Control Interface. To use this part, you need a CD-ROM drive and an MCI CD Audio device driver.

### Operation

To make effective use of this part’s capabilities, you will probably need to create an application with a visual interface, as shown in the Example topic later in this chapter.

The **open** message should be the first message sent to this part, and the **close** message should be the last message sent to it.

### Events

A CD audio part has the following event.

**playCompleted**

Indicates that playing (initiated by a **play**, **playFrom**, or **playFrom:to** message) has completed. Triggered by the MCI device driver when the end of the media is reached.

**Triggered By:** MCI device driver

### Messages

A CD audio part has the following messages.

**audio:** aBoolean

Turn audio output on or off, according to aBoolean.

**Returns:** —

**Triggers:** —

**audioLeft:** aBoolean

Turn left channel audio output on or off, according to aBoolean.

**Returns:** —

**Triggers:** —

**audioRight:** aBoolean

Turn right channel audio output on or off, according to aBoolean.

**Returns:** —

**Triggers:** —
close
Close the receiver.
Returns: —
Triggers: —

closeDoor
Close the door of the CD-ROM drive.
Returns: —
Triggers: —
currentTrack
Answer the current track number.
Returns: an Integer
Triggers: —
currentTrack: trackNumber
Set the current track number to trackNumber.
Returns: —
Triggers: —
isMediaPresent
Answer whether a disc is present in the CD-ROM drive.
Returns: a Boolean
Triggers: —
isNotReady
Answer whether the device state is 'Not Ready'.
Returns: a Boolean
Triggers: —
isDoorOpen
Answer whether the door of the CD-ROM drive is open.
Returns: a Boolean
Triggers: —
isPaused
Answer whether the device state is 'Paused'.
Returns: a Boolean
Triggers: —
isPlaying
Answer whether the device state is 'Playing'.
Returns: a Boolean
Triggers: —
isReady
Answer whether the device state is 'Ready'.
Returns: a Boolean
Triggers: —
isSeeking
Answer whether the device state is 'Seeking'.
Returns: a Boolean
Triggers: —

isStopped
Answer whether the device state is 'Stopped'.
Returns: a Boolean
Triggers: —

isTimeFormatMilliseconds
Answer whether the time format is Milliseconds.
Returns: a Boolean
Triggers: —

isTimeFormatMsf
Answer whether the time format is Minutes/Seconds/ Frames.
Returns: a Boolean
Triggers: —

isTimeFormatTmsf
Answer whether the time format is Tracks/Minutes/ Seconds/Frames.
Returns: a Boolean
Triggers: —

length
Answer the length of the receiver, in the current time format.
Returns: a String, formatted according to the current time format
Triggers: —

lengthOfTrack: trackNumber
Answer the length of the track specified by trackNumber, in units of the current time format.
Returns: a String, formatted according to the current time format
Triggers: —

numberOfTracks
Answer the number of tracks on the current disc.
Returns: an Integer
Triggers: —

open
Open the receiver.
Returns: —
Triggers: —

openDoor
Open the door of the CD-ROM drive (to eject the disc).
Returns: —
Triggers: —
pause
   Pause the receiver.
   Returns: —
   Triggers: —

play
   Begin playing the receiver starting at the current position.
   Returns: —
   Triggers: —

playFrom: startFrame
   Begin playing the receiver starting at the position indicated by startFrame.
   Returns: —
   Triggers: —

playFrom: startFrame to: endFrame
   Begin playing the receiver starting at the position indicated by startFrame, and stop playing when the position reaches endFrame.
   Returns: —
   Triggers: —

position
   Answer the current position, in units of the current time format.
   Returns: a String
   Triggers: —

position: anIntegerOrString
   Set the current position to anIntegerOrString (in units of the current time format). A String is permitted so that non-integer time formats can be specified.
   Returns: —
   Triggers: —

positionOfTrack: trackNumber
   Answer the position of the beginning of the track specified by trackNumber, in units of the current time format.
   Returns: a String
   Triggers: —

reset
   Reset the position to the beginning of the disc.
   Returns: —
   Triggers: —

resume
   Resume playback from the current position, following a pause.
   Returns: —
   Triggers: —
stop
Stop playing.
Returns: —
Triggers: —

**timeFormat**
Answer a String which indicates the current time format.

Returns: a String
Triggers: —

**timeFormat: aTimeFormatString**
Set the current time format to aTimeFormatString.

Returns: —
Triggers: —

**timeFormatMilliseconds**
Set the time format to Milliseconds.

Returns: —
Triggers: —

**timeFormatMsF**
Set the time format to Minutes/Seconds/Frames.

Returns: —
Triggers: —

**timeFormatTmsF**
Set the time format to Tracks/Minutes/Seconds/Frames.

Returns: —
Triggers: —

**Direct-Edit**
Direct-editing the CD audio part lets you change the part name (directly on the workbench).

**Property Edit**
There are no part-specific properties for the CD audio part. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.
Related Parts

- **Wave Audio (Win32 only)**

The wave audio part is a nonvisual part that is used to play wave audio files.

Example

This example demonstrates a visual interface to the CD Audio part's capabilities. Information about the disc and the current track is displayed, and a set of push buttons operate the CD-ROM drive. (You could provide a fancier interface by replacing the labeled push buttons with a toolbar having VCR-style bitmap graphics.)

This example is in the file CDX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)
Check Box

The check box is a visual part that toggles on and off by clicking on the check box. The check box has a Boolean value that is set to true when the box is checked. The check box value can also be toggled by using the `setValue:`, `check`, or `uncheck` messages. A label appears to the right of the check box. It can be set by direct-edit or in the Properties dialog, or it can be changed at runtime with the `setLabel:` message. The initial value for the check box can be set in the Properties dialog.

The following figure illustrates the use of check boxes to select text styles.

![Check Box Example](image)

Operation

There are two styles of check boxes: automatic check box and non-automatic check box.

Clicking on an automatic check box toggles its value and triggers the `clicked:` event with the new value as the event value. Either the `checked` or `unchecked` event is also triggered, depending on the new value.

Clicking on a non-automatic check box triggers the `clicked:` event, but does not toggle the check box value. Non-automatic check boxes allow choices to be validated before they are accepted.

The Boolean value of either kind of check box can be changed with the `setValue:` message. The appearance of the check box changes if necessary to match the new value.
Check Box

Events

checked
Occurs when the user checks the box by clicking or when the box is checked by the setValue: or check messages.

Triggered By: the user
check
setValue:

clicked: aBoolean
Occurs when the user clicks on the check box. If the check box style is automatic, it changes appearance to indicate the new value. The new check box value is provided in aBoolean.

Triggered By: the user
	rightClicked
Occurs when the user clicks on the check box with button 2.

Triggered By: the user

tabbed
Occurs when the user presses the Tab key and the check box has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

unchecked
Occurs when the user unchecks the box by clicking on it or when the box is unchecked by the setValue: or uncheck messages.

Triggered By: the user
uncheck
setValue:

valuesOn
Occurs when the value message is executed and the box is checked.

Triggered By: value

Messages

check
Checks the check box, setting the value to true.

Returns: —
Triggers: checked

disable
Deactivates the check box so that it does not respond to user input. A disabled check box appears with all elements dimmed—the box, the label, and the check mark if the box is checked.

Returns: —
Triggers: —
enable
Enables the check box to respond to user input and restores the original appearance.

Returns: —
Triggers: —

label
Returns the check box label.

Returns: a String
Triggers: —

setFocus
Moves the input focus to the check box. Input focus is indicated by a dashed rectangle around the label. Pressing the Enter key while the input focus is on a check is the same as clicking on it.

Returns: —
Triggers: —

setLabel: aString
Sets the check box label to aString.

Returns: aString
Triggers: —

setValue: aBoolean
Sets the value of the check box to aBoolean—checked if true, unchecked if false, and triggers the corresponding event.

Returns: aBoolean
Triggers: checked unchecked

uncheck
Unchecks the check box, setting the value to false.

Returns: —
Triggers: unchecked

value
Returns the current value of the check box — true if checked, false if not. Triggers valuesOn if check box is checked.

Returns: a Boolean
Triggers: valuesOn

Direct-Edit
Direct-editing the check box allows you to enter or revise the check box label. The label is a text string that is displayed to the right of the box in both the workbench and the runtime application. If the new label is longer than the previous one, the check box changes size so the entire label is visible.
Check Box

Property Edit

Each of the part-specific controls in the dialog is described in the following paragraphs. The Windows version of this part has the additional property of Label left of check box.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

### Label
You can enter the text for the check box label in this field or with direct-edit.

### Styles

- **Auto check box**
  This check box is checked by default. An automatic check box toggles its value when the user clicks on the check box. It triggers the `clicked:` event with the new value and either `checked` or `unchecked`.

  A check box that is not automatic only changes value when the `setValue:`, `check`, or `uncheck` messages are executed. Whether the box appears checked or not depends on the new value. If the value is true, the box appears checked. Non-automatic check boxes respond to mouse or keyboard input with the `clicked:` event, but do not toggle their value or trigger the `checked` or `unchecked` events. A nonautomatic check box can be used when the selection must be verified before it can take place.

- **Label left of check box**
  The default is to place the label to the right of the check box. This property is only available in the Windows version of this part.

### Initially on
The initial value of the check box is determined by this control. The default initial value is off.
Related Parts

- **Radio Button**
  A radio button allows you to choose one alternative from several.

- **Group Pane**
  A group pane can be used to draw a labeled rectangle around a group of related check boxes (or other visual parts).

Remarks

You can define a shortcut key that will click a specific check box when that key is pressed. To define a shortcut key in Windows, the character just before the shortcut character in the label must be the ampersand (&) character. In OS/2, the tilde (~) character is used instead of the ampersand. The character following the ampersand or tilde becomes the shortcut key.

When the label is displayed, the ampersand or tilde is not shown, and the shortcut character is underlined to indicate that it is the shortcut key for that check box. If the focus is on an entry field, you must use Alt plus the shortcut key.

The style of a check box cannot be changed at application runtime.

Example

This example contains two sections. In the upper section, you can press the buttons to send messages to the automatic check box. The results of the messages or clicking on the check box are displayed in the upper entry field. You can change the label on the automatic check box by typing in the lower entry field and pressing Enter.
The check box labeled non-automatic is a check box without the automatic style set. It will not change state if you click on it. It will change state if you click on the automatic check box below it, because this one sends the `setValue:` message to the non-automatic check box.

This example is in the file `CHKBXMPL.PAR` in the `SAMPLE\PARTDEMO` subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The clipboard accessor is a nonvisual part that provides messages to access the system clipboard.

The clipboard accessor allows you to manipulate the contents of the system clipboard. You can copy, cut, and paste Strings and Bitmaps into and out of your application.

Events

**copyException**

Occurs when any of the copy messages fail because the system clipboard can't be opened or the object can't be copied to the clipboard.

*Triggered By:*
*copyBitmap:*  
*copyBitmapFromScreen:*  
*copyString:*

**getException**

Occurs when the messages **getBitmap** or **getString** fail because the system clipboard can't be opened or doesn't contain the requested type of object.

*Triggered By:*
*getBitmap:*  
*getString:

**gotBitmap:** **aBitmap**

Occurs when **getBitmap** succeeds in retrieving a bitmap from the system clipboard.

*Triggered By:*
*getBitmap:

**gotString:** **aString**

Occurs when **getString** succeeds in retrieving a string from the system clipboard.

*Triggered By:*
*getString:

Messages

**copyBitmap:** **aBitmap**

Copies a bitmap to the system clipboard in system-standard bitmap format. The **copyException** event is triggered if the clipboard can't be opened or if the bitmap exceeds 64 KB in size.

*Returns:*

*Triggers:*
*copyException*
Clipboard Accessor

**copyBitmapFromScreen**
Prompts the user to frame a rectangle on the screen and copies that portion of the screen to the system clipboard in system-standard bitmap format. The **copyException** event is triggered if the clipboard can’t be opened or if the bitmap exceeds 64 KB in size.

*Returns:* —
*Triggers:* **copyException**

**copyString: aString**
Copies a string to the system clipboard in system-standard string format. The **copyException** event is triggered if the clipboard can’t be opened or if the string exceeds 64 KB in size.

*Returns:* —
*Triggers:* **copyException**

**getBitmap**
Returns the contents of the system clipboard as a bitmap. Returns **nil** if the clipboard is empty or the contents can’t be accessed as a bitmap. Triggers **gotBitmap:** if successful, **getException** otherwise.

*Returns:* a Bitmap
*Triggers:* **gotBitmap:**
**getException**

**getString**
Returns the contents of the system clipboard as a string. Returns **nil** if the clipboard is empty or the contents can’t be accessed as a string. Triggers **gotString:** if successful, **getException** otherwise.

*Returns:* a String
*Triggers:* **gotString:**
**getException**

**Direct-Edit**

Direct-editing the clipboard changes the part name that appears beneath the icon in the workbench.

**Property Edit**

The clipboard accessor part has no special properties. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties.*
Example

This example lets you enter a string in the top entry field and copy it to the system clipboard by pressing the Copy to clipboard button. The system clipboard contents are pasted into the second entry field when the Paste from clipboard button is pressed. If there isn’t a string in the system clipboard when the paste button is pressed, the `getString` message triggers `getException` which brings up an information dialog.

This example is in the file CLBRXmpl.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The combo box, a visual part, is an entry field that can have its contents set from a drop-down list of suggestions or by typing.

The combo box consists of a text field that is always visible and a drop-down list of suggested entries that can be selected. Choosing an item from the list enters the item text into the text field as if you had typed it. The item text can then be modified as needed.

The following illustration demonstrates a common use for combo boxes — selecting the face and size for a font. The drop down list for the combo box on the right is open:

![Combo Box Illustration](image)

A property can be set which causes the drop-down list to always appear open, in which case the drop-down button does not appear.

**Operation**

To make an entry using a combo box, either:

1. Select the entry field.
2. Type the characters to be entered.
   
   or

1. Click on the down-arrow button to the right of the text field.
2. Make a selection from the displayed list.
3. Modify the item text in the text field by typing from the keyboard, if needed.

To scroll the selection list if a scroll bar is displayed:

> Drag the elevator.

   or

> Click on either of the scroll arrows.

   or

> Use the up and down arrow keys.
Refer to the Entry Field chapter, later in this document, for a complete description of entry field operations.

Events

changed: aString

Occurs when the input focus leaves the combo box and the text in the text field has been changed or when the clear or setValue: messages are executed. The new text is provided in aString.

Triggered By: the user

clear
setValue:

entered: aString

Occurs when the Enter key is pressed and the combo box has focus. The new text is provided in aString.

Triggered By: the user

rightClicked

Occurs when the user clicks button 2 on the combo box.

Triggered By: the user

tabbed

Occurs when the Tab key is pressed and the combo box has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

Messages

clear

Clears the text in the text field.

Returns: —

Triggers: changed:

deleteAll

Removes all items from the selection list.

Returns: —

Triggers: —

deleteItem: aString

Removes the item that matches aString from the selection list.

Returns: —

Triggers: —
disable

Ignores any user input. The down-arrow on a disabled combo box is dimmed.

Returns: —

Triggers: —
enable
Enables the combo box to respond to user input.

Returns: —
Triggers: —

insertItem: aString
Appends aString to the selection list.

Returns: —
Triggers: —

list
Returns the items in the selection list.

Returns: an Array of Strings
Triggers: —

listValue
Returns the currently selected list item (the one displayed in the text field).
Returns nil if the text field contents don’t match any of the list items.

Returns: a String
Triggers: —

setFocus
Moves the input focus to the combo box.

Returns: —
Triggers: —

setList: aString or anArrayOfStrings
Sets the items in the selection list to the items in the argument. Returns the new item list.

Returns: aString or anArrayOfStrings
Triggers: —

setValue: aString
Sets the text in the text field to aString.

Returns: aString
Triggers: changed:

value
Returns the contents of the text field.

Returns: a String
Triggers: —

Direct-Edit
Direct-editing lets you type items into the selection list. Separate the items by pressing the Enter key.
Combo Box

Property Edit

Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

List box always visible

If the check box labelled “List box always visible” is checked, the item list is always displayed and the button does not appear on the combo box. The default is to display the item list only on demand.

Edit contents...

Opens a text pane so you can enter or edit the list of items.

Each line that you type in the text pane becomes an item in the list.
Drag drop...

Opens the following dialog where you can enable the capability for objects to be dragged from and/or dropped on the part:

The Source and the Target check boxes in the Drag group pane are used to enable (or disable) the part as the source and/or the target for a specified set of drag and drop operations.

The Unused and the Chosen and Prioritized list panes in the Drag Target Formats group pane are used to indicate how the part should render a dropped object, and they are disabled unless the part has been enabled as a target. This target part attempts to render the dropped object in one of the chosen formats, beginning with the first entry in the Chosen and Prioritized list pane. If the object cannot be rendered in that format, then the other chosen formats are tried. If the object cannot be rendered in any of the chosen formats, then it is ignored. To move formats between the Unused and the Chosen and Prioritized list panes, drag them from one list to the other. To change the priority order of the chosen formats, drag them within the Chosen and Prioritized list pane.

The Select Operations multiple-selection list pane in the Drag Target Operations group pane is used to indicate the set of drag and drop operations that the part will accept, and it is disabled unless the part has been enabled as a target. An operation is selected if it is highlighted.

The Defaults button is used to restore the part's drag drop property to its default values.

Related Parts

- **Drop-down List**
  A drop-down list looks like a combo box, but only items in the list can be selected.

- **Entry Field**
  An entry field is used to enter a single line of text.
• **List Pane**
  A list pane is a scrollable list of items, only one of which can be selected at a time.

**Remarks**

The size of the combo box, as shown by the selection handles when it is selected, controls how much of the selection list is displayed when the button to the right of the text field is pressed.

To select a combo box in the workbench, click on its entry field.

**Example**

Open the combo box at the upper left of the window and select an item. When the `changed:` or `entered:` event occurs, the event values are displayed in static text parts to the right of the combo box. The `listValue` button sends the `listValue` message to the combo box and displays the result in the adjacent entry field. You can experiment with `listValue` to see that it only returns a string if the contents of the entry field exactly matches one of the list items.

The buttons below the combo box send the messages that appear on their labels. You can add and delete items to the item list by entering them in the appropriate entry field to the right of the buttons.

This example is in the file CMBOXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A comparison part is a nonvisual part that performs basic comparisons.

This part makes comparisons such as greater-than, less-than, and equal-to between two arguments of the same type. A Boolean is returned by each message. In addition, one of two events, **true** or **false**, is triggered by each comparison message.

The relational comparison messages expect values that are magnitudes. Magnitudes include values having the following types:

- Character
- Date
- Number (Integer, Float, or Fraction)
- String
- Time

Values of any type can be tested for equality.

**Events**

**false**

Occurs when the result of a comparison is false.

*Triggered By:* all messages

**true**

Occurs when the result of a comparison is true.

*Triggered By:* all messages

**Messages**

**is: aValue equalTo: anotherValue**

Returns true if the arguments are equal, false otherwise. Either **true** or **false** is triggered based on the outcome of the comparison.

*Returns:* a Boolean

*Triggers:*

- **true**
- **false**

**is: aMagnitude greaterThan: anotherMagnitude**

Returns true if **aMagnitude** is greater than **anotherMagnitude**, false otherwise. Either **true** or **false** is triggered based on the outcome of the comparison.

*Returns:* a Boolean

*Triggers:*

- **true**
- **false**
is: aMagnitude greaterThanOrEqualTo: anotherMagnitude
Returns true if aMagnitude is greater than or equal to anotherMagnitude, false otherwise. Either true or false is triggered based on the outcome of the comparison.

Returns: a Boolean
Triggers: true
false

is: aValue identicalTo: anotherValue
Returns true if the arguments are the same object, false otherwise. Either true or false is triggered based on the outcome of the comparison.

Returns: a Boolean
Triggers: true
false

is: aMagnitude lessThan: anotherMagnitude
Returns true if aMagnitude is less than anotherMagnitude, false otherwise. Either true or false is triggered based on the outcome of the comparison.

Returns: a Boolean
Triggers: true
false

is: aMagnitude lessThanOrEqualTo: anotherMagnitude
Returns true if aMagnitude is less than or equal to anotherMagnitude, false otherwise. Either true or false is triggered based on the outcome of the comparison.

Returns: a Boolean
Triggers: true
false

is: aValue notEqualTo: anotherValue
Returns true if the arguments are not equal, false otherwise. Either true or false is triggered based on the outcome of the comparison.

Returns: a Boolean
Triggers: true
false

is: aValue notIdenticalTo: anotherValue
Returns true if the arguments are not the same object, false otherwise. Either true or false is triggered based on the outcome of the comparison.

Returns: a Boolean
Triggers: true
false

Direct-Edit

Direct-editing the comparison part changes the part name that appears beneath the icon in the workbench.
Property Edit

A comparison part has no special properties. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Example

This example allows you to compare two values as either strings or integers. Each of the six buttons in the lower part of the window triggers the comparison described by its label. All six buttons are linked to one link junction which in turn is linked to the CompareAsInteger comparison part. CompareAsInteger compares the current value of the Integer radio button to the boolean “true” stored in the True value holder.

If the Integer radio button is on (true), the CompareAsInteger part triggers the true event, the contents of the two entry fields are converted to integers, and the results are stored in the Argument1 and Argument2 value holders (Strings that are not valid integers convert to the integer zero). If the radio button is off (false) the CompareAsInteger part triggers the false event and the contents of the entry fields are placed in the Argument value holders without conversion.
Each button also has a second link connected to the Comparison part. Each button sends the message indicated on its label. The Comparison part compares the values in the two Argument value holders. The result of the comparison (true or false) is sent to a static text field for display.

This example is in the file COMPARE.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A computation part is a nonvisual part that performs mathematical computations. The computation part computes addition, subtraction, multiplication, and division between two arguments of type Number, as well as performing logical operations on Boolean arguments. A result of the same type as the arguments is returned by each message.

Numbers include the following types:
- Float
- Fraction
- Integer

Any type of Number is acceptable as an argument for those operations requiring Number arguments.

Events

The computation part does not trigger any events.

Messages

abs: aNumber
Returns the absolute value of aNumber.

Returns: a Number
Triggers:

difference: aNumber and: anotherNumber
Returns the result of subtracting anotherNumber from aNumber.

Returns: a Number
Triggers:

logicalAnd: aBoolean and: anotherBoolean
Returns true if both arguments are true, false otherwise.

Returns: a Boolean
Triggers:

logicalOr: aBoolean and: anotherBoolean
Returns true if either argument is true, false otherwise.

Returns: a Boolean
Triggers:
Computation

**modulo: aNumber and: anotherNumber**
Returns the integer result of dividing `aNumber` by `anotherNumber`. The result is truncated toward negative infinity. `anotherNumber` cannot be zero.

*Returns:* an Integer

*Triggers:* —

**not: aBoolean**
Returns true if `aBoolean` is false, false otherwise.

*Returns:* a Boolean

*Triggers:* —

**product: aNumber and: anotherNumber**
Returns the result of multiplying `aNumber` by `anotherNumber`.

*Returns:* a Number

*Triggers:* —

**quotient: aNumber and: anotherNumber**
Returns the result of dividing the `aNumber` by `anotherNumber`. `anotherNumber` cannot be zero.

*Returns:* a Number

*Triggers:* —

**remainder: aNumber and: anotherNumber**
Returns the integer remainder after dividing `aNumber` by `anotherNumber`. The result is truncated toward negative infinity. `anotherNumber` cannot be zero.

*Returns:* an Integer

*Triggers:* —

**sum: aNumber and: anotherNumber**
Returns the result of adding `aNumber` and `anotherNumber`.

*Returns:* a Number

*Triggers:* —

### Direct-Edit

Direct-editing the computation part changes the part name that appears beneath the icon in the workbench.

### Property Edit

The computation part has no special properties. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties.*
Example

This example lets you enter integer values into two integer entry fields. When a button is pressed, that operation is performed by the computation part and the result is shown in the entry field at the bottom of the window.

This example is in the file COMPUTE.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A confirmer is a predefined two-button dialog—a visual part that presents a two-part message consisting of a title and a message string.

Confirmer-part properties let you set the title, message text, and choose between having Yes and No, OK and Cancel, or Retry and Cancel buttons.

The three types of confirmer parts are shown in the following illustrations:

- **Yes/No confirmer**
  ![Yes/No confirmer](image)
  
  Pressing the Yes, OK, or Retry button triggers the **yes** event. Pressing the No or Cancel button triggers the **no** event.

- **OK/Cancel confirmer**
  ![OK/Cancel confirmer](image)

- **Retry/Cancel confirmer**
  ![Retry/Cancel confirmer](image)

Pressing the Yes, OK, or Retry button triggers the **yes** event. Pressing the No or Cancel button triggers the **no** event.

The confirmer title and message text can be set in the Properties dialog, or at runtime with the `openText:` or `openTitle:text:` messages.
Operation

To operate a confirmer, click on any of the buttons or press the Enter key to select the default button, either Yes, OK, or Retry depending on the style of confirmer.

For the OS/2 version of this part, when the end user’s system warning beep feature is enabled, the speaker beeps each time a confirmation dialog opens. To view the setting of this feature:
1. Open the OS/2 System folder.
2. Open the System Setup folder.
3. From the Sound icon’s pop-up menu, choose Open/Settings.
4. Turn to the Warning Beep page of the settings notebook.
   The warning beep is enabled when the Warning Beep checkbox is checked, and disabled when the checkbox is unchecked.

Events

no
  Occurs when the user clicks on the No or Cancel button.
  Triggered By: the user

yes
  Occurs when the user clicks on the Yes, OK, or Retry button, or the Enter key is pressed.
  Triggered By: the user

Messages

open
  Opens the dialog with the title and message text defined in the Properties dialog.
  Returns: —
  Triggers: —

openText: textString
  Opens the dialog with the title defined in the Properties dialog and the message text contained in textString.
  Returns: —
  Triggers: —

openTitle: titleString text: textString
  Opens the dialog with the title contained in titleString and the message contained in textString.
  Returns: —
  Triggers: —

setText: aString
  Sets the dialog’s text string to aString.
  Returns: —
  Triggers: —
**setTitle: aString**  
Sets the title string of the dialog to *aString*.

*Returns:*  

*Triggers:*  

**text**  
Returns the dialog’s text string.

*Returns:*  
a String  

*Triggers:*  

**title**  
Returns the title string of the dialog.

*Returns:*  
a String  

*Triggers:*  

**Direct-Edit**

Direct-editing the confirmer part changes the part name that appears beneath the icon on the workbench.

**Property Edit**

Each of the part-specific controls in the confirmer’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

**Title**

Enter the title to be displayed in the title bar of the confirmation dialog.

**Text**

Enter the text to be displayed in the confirmation dialog.

**Button labels**

Select which of the three styles of confirmer buttons you want. A Yes/No confirmer is the default.
Confirmer

Wait for answer

The confirmer is an operating-system defined dialog and its Wait for answer property is permanently set. For information on the Wait for answer property, see the Dialog Window chapter later in this document.

Related Parts

- **Confirmer with Cancel**
  The confirmer with cancel has three buttons that let you either accept, reject, or ignore the proposed operation.

Example

This example lets you set the title and text fields and open any of the three styles of confirmers.

This example is in the file CONFIRM.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A confirmer with cancel is a predefined three-button dialog, a visual part that presents a two-part message consisting of a title and a message string.

A confirmer with cancel has a title, a message text, and a choice of either Yes, No, and Cancel buttons or Abort, Retry, and Ignore buttons. Each button triggers the corresponding yes, no, or cancel event.

The two types of confirmer with cancel parts are shown in the following illustrations:

- **Yes/No/Cancel confirmer**

```
? Current application changed, save it?

Yes  No  Cancel
```

- **Abort/Retry/Ignore confirmer**

```
? Continue trying?

Abort  Retry  Ignore
```

The title and message text can be set in the Properties dialog, or at runtime with the openText: or openTitle: messages. You choose the button style in the Properties dialog during development.

**Operation**

To operate a confirmer with cancel, click on any of the three buttons or press the Enter key to select the default button.

For the OS/2 version of this part, when the end user's system warning beep feature is enabled, the speaker beeps each time a confirmation dialog opens. To view the setting of this feature:

1. Open the OS/2 System folder.
2. Open the System Setup folder.
3. From the Sound icon's pop-up menu, choose Open/Settings.
4. Turn to the Warning Beep page of the settings notebook. The warning beep is enabled when the Warning Beep checkbox is checked, and disabled when the checkbox is unchecked.

Events

cancel
Occurs when the user clicks on the Cancel or Ignore button.

Triggered By: the user

no
Occurs when the user clicks on the No or Retry button.

Triggered By: the user

yes
Occurs when the user clicks on the Yes or Abort button, or presses the Enter key.

Triggered By: the user

Messages

open
Opens the dialog with the title and message text defined in the Properties dialog.

Returns: —
Triggers: —

openText: textString
Opens the dialog with the title defined in the Properties dialog and the message text contained in textString.

Returns: —
Triggers: —

openTitle: titleString text: textString
Opens the dialog with the title contained in titleString and the message text contained in textString.

Returns: —
Triggers: —

setText: aString
Sets the dialog’s text string to aString.

Returns: —
Triggers: —

setTitle: aString
Sets the title string of the dialog to aString.

Returns: —
Triggers: —
setWaitForAnswer: aBoolean
If aBoolean is True, this message turns the Wait for answer property (available only in the OS/2 version of this part) on. If aBoolean is False, the message turns the Wait for answer property off.

Returns: aBoolean
Triggers: —

text
Returns the dialog’s text string.

Returns: a String
Triggers: —
title
Returns the title string of the dialog.

Returns: a String
Triggers: —

waitForAnswer
Returns true if the Wait for answer property (available only in the OS/2 version of this part) is turned on, otherwise returns false.

Returns: a Boolean
Triggers: —

Direct-Edit
Direct-editing the confirmer part changes the part name that appears beneath the icon on the workbench.

Property Edit
Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Title
Enter the title to be displayed in the title bar of the confirmation dialog.
Text
Enter the text to be displayed in the confirmation dialog.

Button labels
Select which of the two styles of confirmer buttons you want. A Yes/No/Cancel confirmer is the default.

Wait for answer
Confirmer with cancel is an operating-system defined dialog and its Wait for answer property is permanently set. For information on the Wait for answer property, see the Dialog Window chapter later in this document.

Related Parts
- Confirmer
  The confirmer is a two-button (Yes/No, OK/Cancel, or Retry/Cancel) confirmation dialog.

Example

This example lets you set the title and text fields and open either of the two styles of confirmers with cancel.

This example is in the file CONFIRMC.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Container
(OS/2 only)

A container is a visual part that contains a collection of objects which can be viewed in different formats. The objects can be viewed as full-size icons, miniature icons, or text. They can be placed in arbitrary locations, lined up in columns, arranged in tree form, or shown in a tabular format.

The “Details” view presents the information in the Container in tabular format, which is useful for displaying different attributes of the items. If the Container is not in Details view, the visual presentation of the items is a combination of two attributes: the appearance of Container items, and the placement of Container items.

There are three different ways that Container items can appear:

• In “Icon” view, a Container item appears as an icon/text pair, with the text below the icon.
• In “Name” view, a Container item appears as an icon/text pair, with text to the right of the icon.
• In “Text” view, a Container item appears as a text string.

There are two different features that control the placement of Container items:

• The “Tree” feature arranges the items hierarchically in a tree.
• The “Flow” feature will flow the items from left to right if there are more items than fit vertically in the window.

If neither of these features is set, the standard placement of items is in a single vertical line for “Name” and “Text” views, and wherever the user positions the items for “Icon” view. Note that setting the “Flow” attribute in “Icon” view has no effect.

Operation

Container items can be dragged around within the container, or dragged to new Containers. A common example of a Container is the OS/2 “Drives” icon, which opens up a Container on all drives known to the system.
Events

A container has the following events.

- **aboutToDelete**: `anOrderedCollection`
  Triggered when the user presses the Delete or the Backspace key while one or more Container Items are selected. The argument is an OrderedCollection of the selected items.
  - **Triggered By**: the user

- **changed**: `aContainerItem`
  Triggered when the selection of the Container has changed. The argument is the Container Item most recently selected or deselected.
  - **Triggered By**: the user
deselectAll  
deselectItem:  
deselectItems:  
selectAll  
selectItem:  
selectItems:

- **collapseItem**: `aContainerItem`
  Triggered when an item is collapsed in Tree view. The argument is the Container Item which was collapsed.
  - **Triggered By**: the user
collapseTreeForItem:

delete**: `anOrderedCollection`
  Triggered when one or more items are deleted from the Container. The argument is an OrderedCollection of the deleted items.
  - **Triggered By**: the user
deleteAll  
deleteItem:  
deleteItems:  
deleteSelection:

- **enter**: `aContainerItem`
  Triggered when the user double-clicks on an item, or presses the Enter key while an item is selected.
  - **Triggered By**: the user

- **expandItem**: `aContainerItem`
  Triggered when an item is expanded in Tree view. The argument is the Container Item which was expanded.
  - **Triggered By**: the user
expandTreeForItem:

- **fieldChanged**
  Triggered when the user changes the value of a field (other than the #name field) in Details view by direct editing it. To find out the item, the previous value of the field, and the new value of the field, send the #editData message to the Container.
  - **Triggered By**: the user
getFields
Triggered when the window is opened. If your Container has customized fields for the Details view, you should handle this event by creating a collection of ContainerDetailField objects, then sending this collection to the Container via the #fields: message.

headingChanged
Triggered when the user changes the heading of a column in Details view by direct editing it. To find out the column, its previous heading, and its new heading, send the #editData message to the Container.

Triggered By: the user

insert: anOrderedCollection
Triggered when one or more items are inserted into the Container, either via one of the #insert messages (in the following list) or by dragging an item into the Container and dropping it. The argument is an OrderedCollection of the items which were inserted.

Triggered By: the user
insertFolder:
insertFolder:parent:
insertFolders:
insertFolders:parent:
insertItem:
insertItem:parent:
insertItems:
insertItems:parent:
insertPathName:
insertPathNames:

nameChanged
Triggered when the user changes the name of a Container Item by direct editing it. To find out the item, its previous name, and its new name, send the #editData message to the Container.

Triggered By: the user

needsContents
Triggered when the window is opened. It gives you the opportunity to initialize the contents of the Container.

titleChanged
Triggered when the user changes the title of the Container by direct editing it. To find out its previous title and its new title, send the #editData message to the Container.

Triggered By: the user
A container has the following messages.

**addDragFormat**: *aString*
Adds a drag format to the list of formats recognized by the receiver. The argument is a String representing the drag format.

*Returns*: —
*Triggers*: —

**addMechanism**: *aMechanism format: aFormat*
Add mechanism --> formats to the dictionary of rendering drag/drop mechanisms and formats. If the mechanism is already known, augment its set of formats. The mechanisms and formats are explained in the OS/2 drag/drop documentation (for example, ‘DRM_OS2FILE’ or ‘DRF_TEXT’).

*Returns*: —
*Triggers*: —

**alignTitleCenter**
Centers the Container title.

*Returns*: —
*Triggers*: —

**alignTitleLeft**
Left-aligns the Container title.

*Returns*: —
*Triggers*: —

**alignTitleRight**
Right-aligns the Container title.

*Returns*: —
*Triggers*: —

**arrangeItems**
Arrange the items in the Container. Only applies to Icon view.

*Returns*: —
*Triggers*: —

**changeIconForObjects**: *anOrderedCollection toIcon: anIcon*
Change the icon for a collection of objects. The first argument is the collection of objects in the Container. The second argument is the new icon.

*Returns*: —
*Triggers*: —

**changeIconForPaths**: *anOrderedCollection toIcon: anIcon*
Change the icon for a collection of items. The first argument is the collection of item names in the Container. The second argument is the new icon.

*Returns*: —
*Triggers*: —
**changeNameFor:** *anObject toName: aString*
Change the name of the specified object in the Container.

Returns: —
Triggers: —

**collapseTreeForItem:** *aContainerItem*
Collapses a subtree in Tree view. The argument is the item to collapse.

Returns: —
Triggers:  

**cursoredObject**
Answers the object for the item which most recently received emphasis.

Returns: an Object
Triggers: —

**deemphasizeInUse:** *aContainerItem*
Visually indicate that the specified item is no longer in use by removing the hash marks superimposed on its icon.

Returns: —
Triggers: —

**deleteAll**
Deletes all items from the Container.

Returns: —
Triggers: delete:

**deleteItem:** *aContainerItem*
Delete the specified item from the Container.

Returns: —
Triggers: delete:

**deleteItems:** *anOrderedCollection*
Delete the specified collection of Container items.

Returns: —
Triggers: delete:

**deleteSelection**
Deletes the currently selected items.

Returns: —
Triggers: delete:

**deselectAll**
Deselects all selected items. Does not apply to Containers in single-select mode.

Returns: —
Triggers: changed:

**deselectItem:** *aContainerItem*
Deselects the specified item.

Returns: —
Triggers: changed:
**deselectItems:** anOrderedCollection
Deselects the specified collection of items.

*Returns:* —

*Triggers:* changed:

**disable**
Disables the Container so it does not accept user input.

*Returns:* —

*Triggers:* —

**dragFormats**
Answers a collection of the drag formats recognized by the Container.

*Returns:* a Collection

*Triggers:* —

**editData**
Should be sent to obtain information about a direct-edit action (that is, it should be used in response to the **nameChanged**, **titleChanged**, **headingChanged**, and **fieldChanged** events). This message answers a Dictionary, whose key => values are as follows:

- **oldData** => the old value of what was edited
- **newData** => the new value of what was edited
- **editedItem** => what was edited, as follows:
  - For the **titleChanged** event, the Container.
  - For the **headingChanged** event, the ContainerDetailField.
  - For the **nameChanged** event, the ContainerItem.
  - For the **fieldChanged** event, an (Array with: aContainerDetailField with: aContainerItem).

*Returns:* a Dictionary

*Triggers:* —

**emphasizeInUse:** aContainerItem
Visually indicate that an item is in use by superimposing hash marks over its icon. The argument is the item.

*Returns:* —

*Triggers:* —

**enable**
Enables the Container so it accepts user input.

*Returns:* —

*Triggers:* —

**expandItem:** aContainerItem
Expands a subtree in Tree view. The argument is the item to expand.

*Returns:* —

*Triggers:* expandItem:
**extendedSelect**
Change the Container to extended select mode.

(Returns: —)

(Triggers: —)

**fields: anOrderedCollection**
Set the detail fields for the Container. The argument is a collection of ContainerDetailFields.

(Returns: —)

(Triggers: —)

**firstSelection**
Answers the currently selected item, or the first selected item if more than one item is currently selected.

(Returns: a ContainerItem)

(Triggers: —)

**getIconFor: aContainerItem or anObject**
Answers the icon for the specified Container object.

(Returns: an Icon)

(Triggers: —)

**getItemNamed: aString**
Answers the first Container item having the specified name, if the object is currently inserted into the Container. Otherwise, it will answer nil.

(Returns: a ContainerItem)

(Triggers: —)

**getItemOf: anObject**
Used to convert between Container Items and other Visual Smalltalk objects. The argument is the object. If the object is currently inserted into the Container, this message will answer the Container Item associated with it. Otherwise, it will answer nil.

(Returns: a ContainerItem)

(Triggers: —)

**getNameFor: aContainerItem or anObject**
Answers the name of the specified object in the Container.

(Returns: a Name)

(Triggers: —)

**getObjectOf: aContainerItem**
Used to convert between Container Items and other Visual Smalltalk objects. The argument is a Container Item. If the item is currently inserted into the Container, this message will answer the Visual Smalltalk object associated with it. Otherwise, it will answer nil.

(Returns: an Object)

(Triggers: —)
Container (OS/2 only)

**hideItem**: aContainerItem

Hides the argument.

*Returns:* —

*Triggers:* —

**insertFolder**: aDirectoryOrString

Inserts a folder into the Container. The argument is either a Directory object, or a string representing the full pathName of a valid Directory.

*Returns:* —

*Triggers:* insert:

**insertFolder**: aDirectoryOrString parent: aDirectoryItem

Inserts a folder into the Container under the specified parent. The first argument is either a Directory object, or a string representing the full pathName of a valid Directory. The second argument is the desired parent item.

*Returns:* —

*Triggers:* insert:

**insertFolders**: aCollection

Inserts a collection of folders into the Container. The argument is a collection of valid folders (see #insertFolder: above).

*Returns:* —

*Triggers:* insert:

**insertFolders**: aCollection parent: aFolderItem

Inserts a collection of folders into the Container under the specified parent. The first argument is a collection of valid folders (see above). The second argument is the desired parent item.

*Returns:* —

*Triggers:* insert:

**insertItem**: anObject

Inserts a Visual Smalltalk object into the Container. The argument is any Visual Smalltalk object.

*Returns:* —

*Triggers:* insert:

**insertItem**: anObject parent: anObject

Inserts a Visual Smalltalk object into the Container under the specified parent. The first argument is any Visual Smalltalk object. The second argument is the desired parent item.

*Returns:* —

*Triggers:* insert:

**insertItems**: aCollection

Inserts the specified collection of Visual Smalltalk objects into the Container.

*Returns:* —

*Triggers:* insert:
**insertItems:** \texttt{aCollection parent: anObject}

Inserts a collection of Visual Smalltalk objects into the Container under the specified parent. The first argument is a collection of Visual Smalltalk objects. The second argument is the desired parent item.

- **Returns:** —
- **Triggers:** \texttt{insert:}

**insertPathName:** \texttt{aString}

Inserts a file or folder object into the Container. The argument is a String representing the pathName of a valid file or folder.

- **Returns:** —
- **Triggers:** \texttt{insert:}

**insertPathNames:** \texttt{aCollection}

Inserts a collection of file or folder objects into the Container. The argument is a collection of Strings representing the pathNames of valid files or folders.

- **Returns:** —
- **Triggers:** \texttt{insert:}

**isDetailView**

Answers true if the Container is in Detail view, false otherwise.

- **Returns:** a Boolean
- **Triggers:** —

**isIconView**

Answers true if the Container is in Icon view, false otherwise.

- **Returns:** a Boolean
- **Triggers:** —

**isNameView**

Answers true if the Container is in Name view, false otherwise.

- **Returns:** a Boolean
- **Triggers:** —

**isSelected:** \texttt{aContainerItem}

Answers true if the specified item is selected, false otherwise.

- **Returns:** a Boolean
- **Triggers:** —

**isTextView**

Answers true if the Container is in Text view, false otherwise.

- **Returns:** a Boolean
- **Triggers:** —

**isTreeView**

Answers true if the Container is in Tree view, false otherwise.

- **Returns:** a Boolean
- **Triggers:** —
Container (OS/2 only)

**list**
Answers a collection of the objects currently in the Container.

*Returns:* a Collection

*Triggers:* —

**multipleSelect**
Change the Container to multiple select mode.

*Returns:* —

*Triggers:* —

**objectAt: aPoint**
Answers the object at aPoint, or nil if there is none.

*Returns:* an Object

*Triggers:* —

**objectsMatching: aString**
Answers a collection of objects whose names match aString.

*Returns:* a Collection

*Triggers:* —

**objectUnderCursor**
Answers the object under the cursor, or nil if there is none.

*Returns:* an Object

*Triggers:* —

**selectAll**
Selects all items in the Container. Does not apply to a Container in single select mode.

*Returns:* a Collection

*Triggers:* —

**selectedObjects**
Answers a collection of the items which are currently selected.

*Returns:* a Collection

*Triggers:* —

**selectItem: aContainerItem**
Selects the specified item. If the Container is in single select mode, this deselects the currently selected item.

*Returns:* aContainerItem

*Triggers:* changed:

**selectItems: aCollection**
Selects a group of items. The argument is the collection of items to select. Does not apply to a Container in single select mode.

*Returns:* aCollection

*Triggers:* changed:

**setFocus**
Moves the input focus to the Container.

*Returns:* —

*Triggers:* —
**setList: aCollection**
Sets the items in the Container to the specified collection of objects. If the Container currently has items inserted, these items are deleted.

*Returns:*  

*Triggers:*  

**setListOfPathNames: aCollection**
Performs the same function as `#setList`, but expects its argument to be a collection of valid pathNames (instead of objects).

*Returns:*  

*Triggers:*  

**setValue: aCollection**
Selects the specified collection of items. Any currently selected items are deselected.

*Returns:*  

*Triggers:*  

**singleSelect**
Change the Container to single select mode.

*Returns:*  

*Triggers:*  

**splitColumn**
Answers the number of the column which will appear to the left of the split bar in Details view.

*Returns:*  an Integer  

*Triggers:*  

**splitColumn: anInteger**
Sets the number of the column which will appear to the left of the split bar in Details view.

*Returns:*  

*Triggers:*  

**splitPosition**
Answers the pixel position of the split bar in Details view.

*Returns:*  an Integer  

*Triggers:*  

**splitPosition: anInteger**
Sets the pixel position of the split bar in Details view.

*Returns:*  

*Triggers:*  

**title**
Answers the current title of the Container.

*Returns:*  a String  

*Triggers:*  


title: aString
   Sets the title of the Container.
   Returns: —
   Triggers: —

unhideItem: aContainerItem
   Makes the specified item visible.
   Returns: —
   Triggers: —

viewDetails
   Change the Container to Details view.
   Returns: —
   Triggers: —

viewIcon
   Change the Container to Icon view, leaving the Flowed attribute alone.
   Returns: —
   Triggers: —

viewLarge
   Use large icons in the Container display.
   Returns: —
   Triggers: —

viewMini
   Use small icons in the Container display. Only valid if the “Enable Mini Icons” property is checked.
   Returns: —
   Triggers: —

viewName
   Change the Container to Name view, leaving the Flowed attribute alone.
   Returns: —
   Triggers: —

viewNameFlow
   Change the Container to flowed Name view.
   Returns: —
   Triggers: —

viewNonDetails
   Changes the Container to the non-Details view currently set.
   Returns: —
   Triggers: —

viewNormal
   Resets the Flowed and Tree attributes in the Container view.
   Returns: —
   Triggers: —
viewText
Change the Container to Text view, leaving the Flowed attribute alone.

Returns: —
Triggers: —

viewTextFlow
Change the Container to flowed Text view.

Returns: —
Triggers: —

viewTree
Change the Container to Tree view, keeping the Name/Icon/Text attribute as is.

Returns: —
Triggers: —

viewTreeIcon
Change the Container to tree Icon view.

Returns: —
Triggers: —

viewTreeName
Change the Container to tree Name view.

Returns: —
Triggers: —

viewTreeText
Change the Container to tree Text view.

Returns: —
Triggers: —

Direct-Edit

Direct-edit lets you enter a list of pathnames into the Container. These pathnames can point either to files or directories. Use the Enter key to separate items. Click outside of the Container when you are done entering pathnames. If you have entered a pathname which does not exist, you will be prompted whether to insert the valid pathnames or abort the entire direct edit operation.
Property Edit

The container’s Properties dialog is implemented as a notebook. Its part specific properties are explained in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Selection Style

Container selection styles include single, multiple, and extended select. When in single select mode, exactly one item is selected at all times (similar to a ListBox). When in multiple select mode, zero or more items can be selected at any time, and clicking on an item toggles its selection state (similar to a MultipleChoiceList). When in extended select mode, more than one item can be selected at a time by holding down the control key while clicking on items, and all items can be deselected by clicking the mouse in an area of the Container which has no items.

Read-only text

When this property is on, the user will be unable to direct-edit any items in the Container, and the #nameChanged, #titleChanged, #headingChanged, and #fieldChanged events will never be triggered.

Enable mini icons

When this property is on, the Container will support the use of miniature icons; when the #viewMini message is sent to the Container, all icon views will use miniaturized icons, and the #viewLarge message will restore the icons to full size.

Note that enabling this option increases the memory requirements of your Container application.

Automatically arrange items

When this property is on, the Container will automatically rearrange all items whenever its contents change, through one of the #insert or #delete messages, or when an item is dragged in. If this property is not on, you can use the #arrangeItems message to rearrange the items manually.
**View styles**

Sets the opening view style of the Container. This page presents a visual representation of the different opening view styles. Click on one of these items to set the initial view style. Note that there is no such thing as “Icon/Flow” view, and therefore this particular combination is disabled. The opening view attribute can be overridden by selecting “Open in details view” from the Details page.

**Title**

This text entry field is used to set the initial title of the Container.

**Alignment**

This set of radio buttons is used to select the alignment (either Left, Center, or Right) of the title within the window. The default value is Center.

**Title separator**

This checkbox is used to enable the optional title separator, which is a horizontal line between the title and the Container items. Note that a Container in Details view always has a title separator. Therefore, if your Container is set to open in Details view, the “Title Separator” checkbox will always be checked.
**Open in details view**

This checkbox indicates whether the Container should be opened initially in Details view, which overrides any view style you set from the View page until you send the `#viewNonDetails` message to the Container.

**Split at pixel**

This integer entry field is used to set the initial pixel position at which to place the split bar.

**Columns to left of split bar**

This integer entry field is used to set the initial number of columns to place to the left of the split bar.

**Remarks**

There is a known timing problem having to do with linking any of the `#insert:` messages to either the Window `#opened` event or the Container `#needsContents` event. This timing problem manifests itself as API failures which show up as one of the following walkbacks:

- AllocDetailFieldInfo failed
- InsertDetailField failed
- InsertRecord failed
- InvalidateRecord failed
- SetCnrInfo failed
This timing problem happens sporadically. If you are experiencing one of these problems as the result of performing an `#insert` triggered by one of the two listed events, the suggested workaround is to perform the `#insert` based on another event, one which is triggered later than the Window `#opened` event, such as a button's `clicked` event.

Example

This example demonstrates the capabilities of the container, through a very simple disk browser, with push buttons that enable you to change the views, selection styles, and so on, of the container.

This example is in the file CNRSAMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A conversion part is a nonvisual part that provides messages to perform conversions to and from data types.

The following table shows a partial list of the different classes that the conversion part recognizes and the messages required to convert among them. Empty entries indicate that the conversion isn’t possible.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Output Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>asString:</td>
</tr>
<tr>
<td>Date</td>
<td>asString:   asDate:</td>
</tr>
<tr>
<td>Float</td>
<td>asString:   asInteger: asFloat:</td>
</tr>
<tr>
<td>Integer</td>
<td>asString:   asInteger: asFloat:</td>
</tr>
<tr>
<td>String</td>
<td>asString:   asInteger: asFloat: asDate:</td>
</tr>
<tr>
<td>Time</td>
<td>asString:</td>
</tr>
</tbody>
</table>

**Events**

The conversion part does not have any events.

**Messages**

*asArray: aCollection or aValue*

Returns an array containing either the single element *aValue* or the items in *aCollection*.

*Returns:* an Array

*Triggers:* —

*asDate: aString*

Returns the Date represented by the text in *aString*. The *aString* argument contains a date in any of the following formats:

- 08-28-63
- 28-Aug-63
- August 28, 1963

*Returns:* a Date

*Triggers:* —
Conversion

asFloat: aString or aNumber
Returns the argument converted to a Float. If the argument is aString, it must be represented in one of the following formats:

-123
123.0
123.8e+27

Returns: a Float
Triggers: —

asInteger: aString or aNumber
Returns the argument rounded to an Integer. If the argument is aString, it must be represented in one of the following formats:

123
-123

Returns: an Integer
Triggers: —

asLowerCase: aString
Returns a copy of aString with all upper case characters converted to lower case.

Returns: a String
Triggers: —

asNumber: aString or aNumber
Returns the argument converted to a Number. Numbers can be Integers, Floats, or Fractions. If the argument is aString, it must be in the proper format to represent an Integer or Float.

Returns: a Number
Triggers: —

asString: aValue
Returns aValue converted to a String. Every value has a textual representation.

Returns: a String
Triggers: —

asUpperCase: aString
Returns a copy of aString with all lower case characters converted to upper case.

Returns: a String
Triggers: —

replaceControlCharsIn: aString
Returns aString with all control characters changed to blanks. This changes the argument itself and does not make a copy.

Returns: aString
Triggers: —
sortedAscending: aCollection
Returns a new collection containing the items in aCollection sorted in ascending order.

Returns: a Collection
Triggers: 

sortedDescending: aCollection
Returns a new collection containing the items in aCollection sorted in descending order.

Returns: a Collection
Triggers: 

Direct-Edit
Direct-editing the conversion part changes the part name that appears beneath the icon in the workbench.

Property Edit
The conversion part has no special properties. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Remarks
If a value cannot be converted, nil is returned as the message result.
If a value is contained in a typed value holder, it can be directly converted with the conversion messages of the value holder.
In this example, you enter a string in the first field and press a button. The selected conversion is performed, and the result is displayed in the lower field. In order to display the result as text, the entry field converts the argument of `setValue:` to a string using the equivalent of the `asString` message.

Try the following examples:
- **123** asFloat:
- **abc** asInteger: (non-digits convert to zero)
- **10-3-92** asDate:

This example is in the file CONVERT.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Currency Entry Field

Currency entry fields restrict input to currency related characters and display input using additional formatting characters.

The only characters that you may enter in a currency entry field are the following:

- digits
- minus sign
- decimal point

If you try to enter a different type of character, you hear a beep from the speaker. When the currency entry field loses focus or you change the contents with the setValue: message, the currency entry field displays the input with additional formatting characters such as a currency symbol.

The currency field uses the currency formatting defined by the operating system, which enables you to specify various currency formatting options including the currency symbol.

- In Windows, you can view and edit these settings via the International Settings control panel.
- In OS/2, you can view and edit these settings via the Country object in the System Setup folder.

The value of a currency entry field returned by the value message and set with the setValue: message is a Fraction. The valueFormatted message returns a String that represents the contents as displayed (with formatting characters such as the currency symbol) in the currency entry field.

The value of an empty currency entry field is always nil.

The OS/2 version of this part has an additional Alignment property.

Events

aboutToChangeTo: aValue

Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the clear or setValue: messages are executed. This event is triggered only after aValue has been validated by the entry field's built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the abortChange and retryChange messages to respond to the proposed new value. If this event is triggered by the clear or setValue: messages, you must write a script to perform the validation.
Currency Entry Field

After your validation (if any) is complete, the entry field triggers its `changed:` event with `aValue` as its event value.

**Triggered By:** the user
clear
setValue:

**changed:** `aFraction`

Occurs when the field loses the input focus and the text has changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. The new contents are provided in `aFraction`.

**Triggered By:** the user
clear
setValue:

**entered:** `aFraction`

Occurs when the Enter key is pressed. The contents are provided in `aFraction`.

**Triggered By:** the user

**rightClicked**

Occurs when the user clicks button 2 in the field.

**Triggered By:** the user

**tabbed**

Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

**Triggered By:** the user

**valueException:** `aValue`

Triggered when the argument of the `setValue:` message is not a valid value for the field.

**Triggered By:** `setValue:`

**Messages**

**abortChange**

Rejects the new value entered by the user and resets the entry field’s contents to the original value. This message can only be sent when the user triggers the `aboutToChange:` event. Its behavior is undefined in other situations.

**Returns:** —

**Triggers:** —

**add:** `aNumber`

Returns the result of adding `aNumber` to the value in the field.

**Returns:** a Number

**Triggers:** —

**clear**

Sets the contents to nil.

**Returns:** —

**Triggers:** `changed:`
disable
Ignores user input. The text in a disabled currency entry field is dimmed.

Returns: —
Triggers: —

divide: aNumber
Returns the result of dividing the value in the field by aNumber.

Returns: a Number
Triggers: —

enable
Enables the field to respond to user input.

Returns: —
Triggers: —
multiply: aNumber
Returns the result of multiplying aNumber by the value in the field.

Returns: a Number
Triggers: —

retryChange
Rejects the new value entered by the user and sets the focus back to the
entry field so the user can try again. This message can only be sent when the
user triggers the aboutToChange: event. Its behavior is undefined in other
situations.

Returns: —
Triggers: —

selectAll
Selects all the text in the field.

Returns: —
Triggers: —

setFocus
Moves the input focus to the field. Input focus is indicated by the blinking
text cursor.

Returns: —
Triggers: —

setValue: aValue
Sets the contents of the field to aValue. If aValue is not a Fraction or a String,
it is converted to a String before it is validated. If aValue is not a valid
currency, a warning dialog is displayed and the contents of the field are not
changed.

Returns: —
Triggers: changed:

currencyEntryField.warningDialog:

subtract: aNumber
Returns the result of subtracting aNumber from the value of the field.

Returns: a Number
Triggers: —
Currency Entry Field

**value**

Returns the contents of the field (a Fraction or nil).

*Returns:* a Fraction  
*Triggers:* —

**valueFormatted**

Returns the formatted text string displayed in the field.

*Returns:* a String  
*Triggers:* —

**Direct-Edit**

Direct-editing the currency entry field changes its value. An Invalid input dialog is displayed if you enter characters that are not allowed in the field.

**Property Edit**

The part-specific properties of the currency entry field are described in the following paragraphs. The OS/2 version of this part has an additional Alignment property.

For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

**Contents**

You can change the contents of the currency entry field by either direct-editing or by typing a value in this field.

**Maximum number of digits**

Specifies the maximum number of digits that a user can enter in the field.
Styles

- **Border**
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**
  When checked, the characters in the field can be scrolled left and right if they are not entirely visible within the entry field. If this is not checked, the characters cannot be scrolled. You can continue to enter characters whether or not they are visible. The default is to allow scrolling.

- **Read only**
  When checked, the contents can't be changed by typing. The default is to allow changes.

- **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

- **Alignment**
  For the OS/2 version of this part, the alignment of the text within the formatted entry field can be set to left-aligned, centered, or right-aligned. The default is left-aligned.

Related Parts

- **Alphabetic Entry Field**
  Restricts input to letters.

- **Date Entry Field**
  Restricts input to digits and displays input formatted as a date.

- **Entry Field**
  A general entry field part that allows information to be entered in any format.

- **Float Entry Field**
  Restricts input to floating point numbers.

- **Integer Entry Field**
  Restricts input to integer numbers.

- **Phone Number Entry Field**
  Restricts input to digits and displays input in phone number format.

- **Picture Entry Field**
  Restricts input to a pattern specified in the field's properties dialog.

Example

FORMAT.PAR, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory, demonstrates all of the formatted entry fields. It is described in the *Alphabetic Entry Field* chapter earlier in this document.
Date Entry Field

Date entry fields are visual parts that operate like entry fields, except that they restrict input to digits and display their contents in date format.

The date entry field accepts either six or eight digits:

013194
01311994

If you try to enter a non-digit character, your system speaker beeps. If you do not enter exactly six or eight digits specifying a valid date, a confirmer is displayed when the field loses focus or you use the `setValue:` message.

Clicking on Yes returns the input focus to the field so you can correct your entry. Clicking on No restores the contents to the previous value.

When the field loses focus or you change the contents with the `setValue:` message, the contents are displayed with date separators. For example, the entry 013194 becomes 01/31/94.

By default, the date entry field uses the date formatting defined by the operating system, which enables you to specify various date formatting options including the separator symbol.

- In Windows, you can view and edit these short date settings via the International Settings control panel.
- In OS/2, you can view and edit these date settings via the Country object in the System Setup folder.

The value of a date entry field returned by the `value` message and set with the `setValue:` message is a Date. The `valueFormatted` message returns a String that represents the contents as displayed (with formatting characters such as the separators).

The value of an empty date entry field is always nil.
Events

aboutToChangeTo: aValue
Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the clear or setValue: messages are executed. This event is triggered only after aValue has been validated by the entry field’s built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the abortChange and retryChange messages to respond to the proposed new value. If this event is triggered by the clear or setValue: messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its changed: event with aValue as its event value.

Triggered By: the user
    clear
    setValue:

changed: aDate
Occurs when the field loses the input focus and the text has changed since the input focus entered this field, or when the clear or setValue: messages are executed. The new contents are provided in aDate.

Triggered By: the user
    clear
    setValue:

entered: aDate
Occurs when the Enter key is pressed. The contents are provided in aDate.

Triggered By: the user

rightClicked
Occurs when the user clicks button 2 in the entry field.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

valueException: aValue
Triggered when the argument of the setValue: message is not a valid value for the field.

Triggered By: setValue:
Messages

abortChange
Rejects the new value entered by the user and resets the entry field's contents to the original value. This message can only be sent when the user triggers the **aboutToChange:** event. Its behavior is undefined in other situations.

Returns:  
Triggers:  

clear
Sets the contents to nil.

Returns:  
Triggers:  

disable
Ignores user input. The text in a disabled date entry field is dimmed.

Returns:  
Triggers:  

enable
Enables the field to respond to user input.

Returns:  
Triggers:  

retryChange
Rejects the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the **aboutToChange:** event. Its behavior is undefined in other situations.

Returns:  
Triggers:  

selectAll
Selects all the text in the field.

Returns:  
Triggers:  

setFocus
Moves the input focus to the field. Input focus is indicated by the blinking text cursor.

Returns:  
Triggers:  

setValue: aValue
Sets the contents of the field to aValue. If aValue is not a Date or a String, it is converted to a String before it is validated. If aValue is not a valid date, a warning dialog is displayed and the contents of the field are not changed.

Returns:  
Triggers:  

changed:
valueException:
value

Returns the contents of the field as a Date or nil.

Returns: a Date or nil
Triggers: —

valueFormatted

Returns the formatted text string displayed in the field.

Returns: a String
Triggers: —

Direct-Edit

Direct-editing a date entry field changes its value. If you enter characters that are not allowed in the field, you see an Invalid input information dialog.

Property Edit

The part-specific properties of the date entry field are described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Contents

You can change the contents of the date entry field by either direct-editing or by typing a value in this field.
Styles

- **Border**  
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**  
  When checked, the characters in the field can be scrolled left and right if they are not entirely visible within the entry field. If this is not checked, the characters cannot be scrolled. You can still enter text that is longer than the field, but you cannot scroll it. The default is to allow scrolling.

- **Read only**  
  When checked, the contents can’t be changed by typing. The default is to allow changes.

- **Password**  
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

Format

- **Use system settings**  
  Enables you to either use the date format settings from the system (when checked) or override them for this particular date entry field.

- **Order:**  
  When the Use system settings check box is turned off, three radio buttons become enabled. These radio buttons allow you to choose one of the following date formats: 'D-M-Y', 'M-D-Y', or 'Y-M-D'.

- **Year:**  
  A pair of radio buttons enable you to choose between a Two digit year and a Four digit year, giving you the option of displaying the century digits.

- **Separator:**  
  When the Use system settings check box is turned off, this entry field becomes enabled. It enables you to type a single character which becomes the date separator.

- **Sample:**  
  Displays the date as it appears using the current settings.

Related Parts

- **Alphabetic Entry Field**  
  Restricts input to letters.

- **Currency Entry Field**  
  Restricts input to currency related characters and displays input formatted as currency.

- **Entry Field**  
  A general entry field part that allows information to be entered in any format.

- **Float Entry Field**  
  Restricts input to floating point numbers.
• **Integer Entry Field**
  Restricts input to integer numbers.

• **Phone Number Entry Field**
  Restricts input to digits and displays input in phone number format.

• **Picture Entry Field**
  Restricts input to a pattern specified in the field’s properties dialog.

**Example**

FORMAT.PAR, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory, demonstrates all of the formatted entry fields. It is described in the *Alphabetic Entry Field* chapter earlier in this document.
The DDE client part, a nonvisual part, allows access to DDE server applications through the use of Dynamic Data Exchange (DDE).

A DDE exchange is a conversation between a client and a server. The client requests something from the server and then the server responds to the request. The DDE Client part allows you to build a client application. The server could be another PARTS Workbench application, a custom application written in another language, or an "off-the-shelf" application that has been programmed to act as a DDE server.

**Operation**

You specify both the name of the server application and the topic of communication in the DDE client part's Properties dialog. The server application must be running before the client sends it a message.

The DDE client can send instructions and data to a server and receive instructions and data from a server. To send information to a server, the DDE client sends a poke-item message. The server, in turn, must have a corresponding poke-item event which is triggered by the poke-item message.

A DDE server sends information and data to a client by sending a hot-link item message. The client, in turn, must have a corresponding hot-link item event which is triggered by the hot-link item message. Both kinds of messages can have arguments and both kinds of events can have values. You'll notice that these messages and events seem very similar to the messages and events that you have encountered elsewhere in PARTS Workbench. There are, however, some important differences.

Poke-item messages and hot-link item messages are transmitted from application to application by the operating system rather than PARTS Workbench. You don't create explicit visual links from one application to another. This means that your PARTS Workbench application can communicate with any application that supports DDE communications—not just other PARTS Workbench applications. Poke-item and hot-link item events and messages are also defined differently than other events and messages.

Poke-item messages and hot-link item events are specified in the DDE client part's Properties dialog. For each poke-item message you define, a new message with the same name and one argument is added to the DDE client part. For each hot-link item event you define, a new event having the same name and one value is added to the DDE client. These events are triggered when the server executes the corresponding message.
Events

The DDE client part has events with the same names as any hot-link item events that you specify in the Properties dialog. Each event has a single argument. When the item is updated, the event is triggered with the new value as the event value.

**initiateException:** applicationString topic: topicString

Occurs when the conversation can’t be initiated. The application and topic values set in the Properties dialog are provided as event values.

*Triggered By:* initiate

**hotLink:** aValue

Occurs when the server updates one of the hot-linked items defined in the Properties dialog. The new item value is provided by aValue.

*Triggered By:* the server hotLink message

Messages

The DDE client part has messages with the same names as any poke-items you specify in the Properties dialog. Each message has a single argument. To update an item, execute its message with the new value as the argument.

**executeCommand:** aString

Requests that the server execute the command aString.

Returns: —

Triggers: the server executeCommand: event

**initiate**

Initiates the DDE conversation. This message must be executed before any of the others.

Returns: —

Triggers: —

**pokeItem:** aString value: aValue

Sends aValue to the server for the item named aString.

Returns: —

Triggers: the server pokeItem:value: event

**pokeItem:** aValue

Sends aValue to the server as a new value for the item named by the message.

Returns: —

Triggers: the server pokeItem event

**requestItem:** aString

Obtains the value of the item named aString from the server.

Returns: a Value

Triggers: —

**terminate**

Terminates the DDE conversation.

Returns: —

Triggers: —
Direct-Edit

Direct-editing the DDE client part changes the part name that appears beneath the icon in the workbench.

Property Edit

Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Application

The name of the server application. A string must be entered in this field.

Topic

The topic for the DDE conversation. A string must be entered in this field.

Hot link item events

Hot-link item events are triggered by hot-link item messages from the server. When you define a hot-link item event, an event of the same name with one value is added to the DDE client part. When the server executes a hot-link item message, the event having the same name is triggered and provides the hot-link item message argument as a value.

To add a new hot-link item event:

1. Enter its name in the entry field.
2. Press the Add button.
To delete a hot-link item event:
1. Select it in the list pane.
2. Press the Remove button.

Poke item messages
Poke-item messages are used to send instructions and data to the server. When you define a poke-item message, a message with the same name and one argument is added to the DDE client. To send a new value to the server, execute the message having the same name as the poke-item message and supply the new value as the message argument.

To add a new poke-item message:
1. Enter its name in the entry field.
2. Press the Add button.

To delete a poke-item message:
1. Select it in the list pane.
2. Press the Remove button.

Related Parts
- **DDE Server**
  Allows a PARTS Workbench application to act as a DDE server.

Remarks
You must successfully initiate a conversation before you can make requests.

When you add poke items or hot-link items, the corresponding message or event is created with a colon appended to indicate that it takes an argument or provides a value. Therefore, do not include a colon in a poke item or hot-link item name; otherwise, the resulting message or event is created with a bogus argument or event value connection.
Example

This example is a single application containing two windows (simulated applications) that communicate with each other via DDE. The client and server have been placed in a single application for this demonstration to guarantee that the applications are opened and that communications are initiated in the proper order (discussed below). Ordinarily, the server and the client are two completely separate applications.

The server window has a single text pane in it and is meant to simulate a commercial word processor. The other window is considered the client window. The user can type a word into an entry field in the client window, press the Count button, and the server window will be asked (via DDE) how many occurrences of that word it contains. The server window computes the answer and sends it back to the client window, which updates itself to display the answer.

The DDE server must be given a name and a topic which it will use to register itself with the operating system. The DDE client must also be given these same names so it can tell the operating system which application it wants to connect to. The names are arbitrary. All that matters is that the client and server both use the same names. If you were building an application to communicate with an external (non-PARTS Workbench) application, you would need to know the names used by that external application and configure the DDE client or server parts accordingly. The application name and the topic of both DDE client and DDE server are set in the Properties dialogs.

In the example application, the DDE client has a poke-item message named `searchWord:` while the DDE server has the poke-item event `searchWord:`. The DDE server has the hot-link item message named `searchCount:` while the DDE
client has the hot-link item event searchCount: (A PARTS Workbench application that dealt with an external DDE client or server would use a DDE part extended with respect to the external application's DDE poke item and hot-link names.)

When the application is launched, the following initialization actions occur:

- The server window is opened by the application's open event.
- The server window's opened event executes DDEServer1's initiate message.
- The result link executes the client window's open message.
- The client window's opened event invokes DDEClient1's initiate message.

**NOTE:** DDE servers must be running and registered with the operating system before a DDE client can connect to them. Invoking DDEClient1's initiate message as an indirect result of DDEServer1's initiate message ensures this.

At this point, both windows are open, DDEServer1 is registered with the operating system (by its initiate message), and DDEClient1 has connected with DDEServer1 (by DDEClient1's initiate message). Now you can type something in the text pane in the server window, and then type in a word in the entry field in the client window. When you press the Count button in the client window, the following actions occur:

- The button's clicked event invokes DDEClient1's searchWord: message.
- The argument for the searchWord: message is the result of executing EntryField1's value message.
- Executing the searchWord: message causes a DDE action by the operating system - an item named searchWord with a value of the contents of EntryField1 will be poked to the server.

Execution continues on the other side of the exchange:

- DDEServer1's searchWord: event invokes a one-line script attached to the text pane named OccurrencesOfWord: This message has a result which is the number of times the argument word appears in the contents of the text pane. You can examine this script by opening the example file and asking for the interface of the text pane.
- The result of OccurrencesOfWord: is forwarded with a result link to DDEServer1's searchCount: message.

Again, the operating system takes over to make the exchange. PARTS Workbench execution picks up below:

- DDEClient1's searchCount: event invokes StaticText2's setValue: message.

This ends the cycle that is executed each time the user clicks on Count.

The only other links in the application are for terminating the DDE parts when the windows are closed.

This example is in the file DDEXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The DDE server part, a nonvisual part, allows access to DDE client applications through the use of Dynamic Data Exchange (DDE).

A DDE exchange is a conversation between a client and a server. The client requests something from the server and then the server responds to the request. The DDE server part allows you to build a server application. The client could be another PARTS Workbench application, a custom application written in another language, or an “off-the-shelf” application that has been programmed to act as a DDE client.

Operation

You specify both the name by which clients will call the server application and a topic of communication in the DDE server part’s Properties dialog. The server application must be running before the client sends it a message.

The DDE server can send instructions and data to a client and receive instructions and data from a client. To send information to a client, the DDE server sends a hot-link item message. The client, in turn, must have a corresponding hot-link item event which is triggered by the hot-link item message.

A DDE client sends information and data to a server by sending a poke-item message. The server, in turn, must have a corresponding poke-item event which is triggered by the poke-item message. Both kinds of messages can have arguments and both kinds of events can have values. You’ll notice in fact that these messages and events seem very similar to the messages and events that you have encountered elsewhere in PARTS Workbench. There are, however, some important differences.

Poke-item messages and hot-link item messages are transmitted from application to application by the operating system rather than PARTS Workbench. You don’t create explicit visual links from one application to another. This means that your PARTS Workbench application can communicate with any application that supports DDE communications—not just other PARTS Workbench applications. Poke-item and hot-link item events and messages are also defined differently than other events and messages.

Hot-link item messages and poke-item events are specified in the DDE server part’s Properties dialog. For each hot-link item message you define, a new message with the same name and one argument is added to the DDE server part. For each poke-item event you define, a new event having the same name and one value is added to the DDE server. These events are triggered when the client executes the corresponding message.
Events

The DDE server has events with the same names as any poke-items you specify in the Properties dialog. Each event has a single argument. When the item is received from the client, the event is triggered with the new value as the event value.

**executeCommand:** `commandString`

Occurs when a client executes the `executeCommand:` message. The event value is the command string to be executed.

*Triggered By:* DDE client `executeCommand:` message

**pokeItem:** `itemString value: valueString`

Occurs when the client executes the `pokeItem:value:` message. The item and value provided are those passed by the client.

*Triggered By:* DDE client `pokeItem:value:` message

**pokeItem:** `aValue`

Occurs when the client executes one of its poke-item messages. The new item value is provided by `aValue`.

*Triggered By:* DDE client `pokeItem` message

Messages

The DDE server part has messages with the same names as any hot-linked items you specify in the Properties dialog. Each message has a single argument. To send updated item values to a client, execute the corresponding hot-link message with the new value as the argument.

**executionFailed**

When executed as the result of the `executeCommand:` event, notifies the operating system that the command execution was unsuccessful.

*Returns:* —

*Triggers:* —

**hotLink:** `aValue`

Triggers the clients corresponding hot-link event with `aValue` as its value.

*Returns:* —

*Triggers:* DDE client `hotLink:` event

**initiate**

Registers the server with the operating system so it can accept DDE requests from client applications.

*Returns:* —

*Triggers:* —

**pokeFailed**

Notifies the operating system that a poke operation was unsuccessful.

*Returns:* —

*Triggers:* —
terminate
Terminates all DDE conversations with clients.

Returns: —
Triggers: —

Direct-Edit
Direct-editing the DDE server changes the part name that appears beneath the icon in the workbench.

Property Edit
Each of the part-specific properties is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Application
The name of the application the server represents. A string must be entered in this field.

Topic
The name of the topic the server recognizes.

Poke item events
Poke-item events are triggered when the client executes the corresponding poke-item message. When you define a poke item event, an event of the same name with one value is added to the DDE server.

To add a new poke-item event:
1 Enter its name in the entry field.
2 Press the Add button.

To delete a poke-item event:
1 Select it in the list pane.
2 Press the Remove button.

Hot link item messages
Hot-link items send instructions and data to the client. When you define a hot-link item message, a message of the same name with one argument is added to the DDE server. When the server executes the message, the event having the same name is triggered in the DDE client.

To add a new hot-link item message:
1 Enter its name in the entry field.
2 Press the Add button.

To delete a hot-link item message:
1 Select it in the list pane.
2 Press the Remove button.

Related Parts
• DDE Client
  The DDE client part lets a PARTS Workbench application be a client of a DDE server.

Remarks
When you add poke items or hot-link items, the corresponding message or event is created with a colon appended to indicate that it takes an argument or provides a value. Therefore, do not include a colon in a poke item or hot-link item name; otherwise, the resulting message or event is created with a bogus argument or event value connection.

The DDE server must be initiated and registered (via the initiate message) before a client can initiate with it.

Example
See the example in the DDE Client chapter earlier in this document.
Dial Pane

The dial pane, a visual part, is a gauge with an indicator that points to an integer value.

The value of a dial pane is changed by dragging the indicator or clicking on the new value. When you resize the dial pane part, the circular dial is automatically resized to fit within the rectangular border.

Whenever the value of the dial pane changes, the `changed:` event is triggered with the new value as the event value. The `setValue:` message allows you to supply a new value for the dial pane. The dial indicator always indicates the current value of the dial pane.

The OS/2 version of this part enables you to set the initial value by dragging the indicator while in direct-edit mode.

Properties let you set the range of values, the number of tick marks, how the tick marks are sized and labeled, and what part of a full circle the scale spans.

The following illustration shows the use of two dial panes to represent an indoor/outdoor thermometer:

```
Operation
To change the value of the dial pane:
> Drag the indicator to the new value.

or

> Click on the desired value on the perimeter of the dial.
```
Dial Pane

Events

changed: anInteger
Occurs whenever the value is changed, either with the indicator or the
setValue: message. The event value is the new dial value.

Triggered By: the user
setValue:

rightClicked
Occurs when the user clicks mouse button 2 on the dial pane.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the dial pane has the input focus.
You can connect this event to the setFocus message of another part to set
the tab order.

Triggered By: the user

Messages

disable
Causes the dial pane to ignore user input.

Returns: 
Triggers: 

enable
Allows the dial pane to respond to user input.

Returns: 
Triggers: 

setFocus
Moves the input focus to the dial pane.

Returns: 
Triggers: 

setValue: anInteger
Sets the value of the dial pane. If anInteger is outside of the range of values,
the value is set to the closest value in the dial pane range.

Returns: 
Triggers: changed:

value
Returns the current value of the dial pane.

Returns: an Integer
Triggers: 

Direct-Edit

There is no direct-editing for the Windows version of the dial pane.

For the OS/2 version of the dial pane, direct-editing enables you to set the initial
value by dragging the indicator.
Property Edit

Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Styles

If Border is checked, a rectangular border is drawn around the dial pane.

Edit contents...

Pressing this button brings up the dial pane Edit dialog. This dialog lets you set the range of values, the portion of a full circle spanned by the scale, the number of increments (tick marks), their size, an initial value, and default and custom labeling for the tick marks.
Dial Pane

- **Range**
  Enter the minimum and maximum values for the dial pane scale. Range values increase in a clockwise direction beginning at the minimum range angle. The maximum value must be greater than the minimum value. The values may be negative or positive.

- **Range angles**
  Range angle settings control how the range values are displayed. These numbers indicate the position on the circle where the minimum and maximum range values are placed. Numbering begins with 0 on the right and increases in a counter-clockwise direction to 90 (straight up), to 180 (on the left), 270 (straight down), and 360 (back pointing to the right).

The following illustration shows the appearance of a dial pane using the default range angle settings on the left and two variants:

- **Number of ticks**
  Sets the number of tick marks that appear on the scale. The number should usually be an integer factor of the range plus one. For example, if the range is from 0 to 100, you could specify 101 ticks, for a tick mark indicating each number, or 51 tick marks for a tick mark every other number, or 26 tick marks for a tick mark every fourth number, and so on.

- **Default tick length**
  Sets the length in pels for tick marks that do not have their tick length modified by other properties. The default length is 5.

- **Initial setting**
  Sets the initial value for the dial pane. The initial value must be in the value range. The default initial value is the minimum value.
• **Modify ticks by interval**
  Lets you specify that the ticks which are Tick interval apart should have a different length. The default is that every tenth tick has a Tick length of 9 pels. If Auto labels is checked (the default), then the modified ticks are labeled with their value.

• **Modify individual ticks**
  Lets you specify a length and label for individual tick marks.

  **To modify an individual tick mark:**
  1. Enter the number of the tick to be modified in the Tick drop down list.
     The tick number is the number of the tick mark, counting clockwise, not the tick value.
  2. Enter the length in pels for this tick mark into the Length field.
  3. Enter the label for this tick mark in the Text field.
  4. Repeat steps 1 through 3 for any additional tick marks you want to modify.

**Example**

The example allows you to enable the dial pane for user input and disable it. When it is enabled you can use the mouse to move the needle to the desired position on the dial or you can move the box in the horizontal scroll bar to adjust the dials setting. When the dial pane is disabled you can only set it with the scroll bar. The numeric value of the dial pane setting is displayed in the entry field to the right of the dial pane.
The Windows version of this example is in the file DIALXMPW.PAR, and the OS/2 version is in the file DIALXMO.PAR, in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Dialog Window

The dialog window, a visual part, is a pop-up window that contains other visual parts.

A dialog window part looks much like a window part and has some of the same controls. It is intended to be used whenever a specific interaction is required and closed when the interaction is complete. It cannot be resized.

Operation

A dialog window is modal; you cannot interact with the window that opened the dialog window until you close the dialog window. Other windows on the desktop can still be used.

For an example of a dialog window, look at any old-style Properties dialog, such as the one for the dialog window itself:

Dialog window properties let you control the appearance of the border, the controls to be displayed at the top of the window, and whether or not the dialog window can be maximized or minimized. You can also set the Wait for answer property.

The behavior of the open message of the dialog window part depends on whether the Wait for answer property has been set. If it has been set, the open message will not return until the dialog window is closed. If this property is unchecked, the open message will return as soon as the dialog window has finished opening.

Setting this property has two implications: First, the processing of the event which lead to the firing of the open message stays on hold until the dialog window is closed. So, if there are further links from the event which lead to the open message,
Dialog Window

they will wait until the open message finishes. Second, if this property is set, and the open message of the dialog window has a result link, the result link will not execute until the window is closed. For example, you cannot use a result link from the open message to initialize the contents of the dialog window. By the time the initialization link executes, the dialog window will be closed.

Typically, you should leave this property unchecked, and ignore the result of the open message. If you have a pair of OK and Cancel buttons in your dialog window, you can create links from the clicked event of these buttons to process responses from the dialog window.

There are some cases, however, which you do need to set the Wait for answer property. One example is when you want to enable the user to verify that he or she wants to close the window without saving the contents (or perhaps give him or her the chance to save it).

The window part has an event called aboutToClose. If you want to veto the closing of the window, you need to send a message called abortClose to the window before the processing of the aboutToClose event is over.

To create an example that requires the Wait for answer property to be set:

1. Drag a confirm from the PARTS Catalog to the Workbench.
2. Double-click on the confirm to open its Properties dialog.
3. Verify that the Wait for answer property is checked.

   In Windows, the confirm is a standard dialog and the Wait for answer property is permanently set.

4. Enter the following in the Text: entry field: “Are you sure you want to close the window?”
5. Click on the OK button to close the Properties dialog.
6. Drag a window from the PARTS Catalog and drop it next to the confirm.
   A link is automatically created from the application’s open event to the window’s open message.
7. Create a link from the window’s aboutToClose event to the confirm’s open message.
8. Create a link from the confirm’s no event to the window’s abortClose message.
9. Click the Launch button located on the Workbench’s tool bar.
   The window is opened.
10. Double-click on the window’s title-bar icon (in OS/2) or control-menu box (in Windows) to close the window.
   The confirm is displayed and execution of the link sending the open message is suspended.
11. Click on the No button.

   The confirm’s no event is triggered and the abortClose message is sent to the window. After the message is sent, the confirm closes and execution of the link that opened the confirm is resumed. Since the confirm sent the
**Abort Close** message to the window while the link was suspended, the window is not closed.

12 Double-click on the window's title-bar icon (in OS/2) or control-menu box (in Windows) again.

13 Click on the Yes button this time.

There is no link fired by the **yes** message so the confirmer simply closes. Execution of the link that opened the confirmer is resumed: the **open** message returns and the **aboutToClose** event continues executing. Since the **abortClose** message was not sent to the window, it is closed.

Another consideration involves the dialog window's **close** message. In the application illustrated below, there are two links connected to the OK button's **clicked** event. One link sends the close message to the dialog window while the other puts the ID Code entered in the dialog window into the Result window's entry field.

We can examine the firing sequence of these two links by choosing the **Selected/Sequence links...** menu item. This dialog shows us that the **close** message is executed first followed by the **setValue:** message.

If the dialog window's Wait for answer property is not set, the application will perform as expected:

1. The user enters an ID Code.
2. The user clicks on the OK button.
3. The dialog window closes.
4. The **setValue:** message retrieves the ID Code with an argument link and places it in the Result window's entry field.

If, however, the Wait for answer property is set, the dialog will suspend execution of the link sending the **close** message and then close itself without ever having signaled execution to resume. In this case, the ID Code entered in the dialog box would not appear in the Result window's entry field.

This problem is easily solved by using the Sequence Links dialog to make the **setValue** message execute before the **close** message.
NOTE: The confirmers, confirmers with cancel, information dialog, and file dialog are implemented by Windows, and due to a restriction in Windows, the Wait for answer property cannot be turned off. If you build your own dialog using the dialog window part, you can of course set this property as you wish.

Events

aboutToClose
Occurs when the close message is sent or the user closes the window. If this event triggers the execution of the abortClose message, the window does not close.

Triggered By: the user
close

closed
Occurs just after the window closes.

Triggered By: the user
close

opened
Occurs just after the window is opened.

Triggered By: open

rightClicked
Occurs when the user clicks mouse button 2 on the dialog window.

Triggered By: the user

Messages

abortClose
If this message is executed as a result of triggering the aboutToClose event, the window does not close.

Returns: —
Triggers: —

close
Triggers the aboutToClose event. If the abortClose message is executed as a result of triggering aboutToClose the window does not close. Otherwise the window triggers the closed event and closes.

Returns: —
Triggers: aboutToClose
closed

label
Returns the text in the title bar.

Returns: a String
Triggers: —
**open**
Opens up the new dialog window and suspends the execution of the window containing the part sending the `open` message.

- **Returns:**
  -
- **Triggers:** `opened`

**setLabel: aString**
Sets the title in the title bar to the argument.

- **Returns:** `aString`
- **Triggers:**
  -

**setWaitForAnswer: aBoolean**
If `aBoolean` is True, this message turns the Wait for answer property on. If `aBoolean` is False, the message turns the Wait for answer property off.

- **Returns:** `aBoolean`
- **Triggers:**
  -

**waitForAnswer**
Returns true if the Wait for answer property is set, otherwise returns false.

- **Returns:** a Boolean
- **Triggers:**
  -

**Direct-Edit**
Direct-editing a dialog window allows you to type a new label into the title bar.

**Property Edit**
Each of the part-specific controls in the dialog window’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Dialog Window

Dialog label
You can enter the text that appears in the title bar in this field. The default is the part name.

Styles
For an explanation of what aspects of the window's appearance are being controlled, see the Window chapter later in this document.

Check boxes allow you to turn the following dialog window features on or off:

- **Title bar**
  Controls whether the title bar containing the title is displayed at the top of the window. The title bar is displayed by default.

- **Title-bar icon (OS/2) or Control-menu box (Windows)**
  Controls whether to display the title-bar icon (in OS/2) or the control-menu box (in Windows) in the upper left corner of the window. It is displayed by default.

Border
You can select one of two different border styles. The default is dialog border.

- **Dialog border**
  Displays a heavy border around the dialog window as shown in the following illustration. This is the default border style.

- **Single-line border**
  The single line border appears as shown in the following illustration. The single-line border is typically used without the title bar and title-bar icon (in OS/2) or control-menu box (in Windows).
**Wait for answer**

If this property is checked, the dialog window's open message will not return until the dialog window is closed. If this property is unchecked, the open message will return as soon as the dialog window has finished opening. See Operation earlier in this chapter for a more complete discussion of this property.

**Opening styles...**

This button opens the following dialog that allows the user to choose the opening state of the dialog window.

The Initial position after opening can be either Relative to the display or Centered on the cursor. Relative to the display uses the position of the dialog window in relation to the Workbench, and during runtime, positions the dialog window in the same relation to the display. Centered on the cursor displays the dialog window at the cursor position when opened. Relative to the display is the default setting.

The OS/2 version of this part has the additional property of Open as desktop child. When this property is checked, you can place the dialog window within another window yet have it remain a child of the desktop so you can move it outside of the other window.

**Related Parts**

- **Default Push Button**
  
  A push button in a dialog window that has the default button style set is clicked when the Enter key is pressed and the input focus is on the dialog window.
This example allows you to open a dialog window, change the label in its title bar, and close it. The `opened` and `closed` events triggered by the dialog window are indicated in the two entry fields at the bottom of the window.

This example is in the file DIALOGW1.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
**Dictionary Holder**

A dictionary holder is a nonvisual part that serves as a variable to hold a Dictionary. The dictionary holder is a value holder with the class set to Dictionary so that additional Dictionary-specific messages are available.

See the *Value Holder* chapter later in this document for information about the events and messages common to all value holders.

**Operation**

A Dictionary is a data structure that stores and retrieves values using a key. The Dictionary messages:

- **at: aKey**
- **at: aKey put: aValue**

access and store the value corresponding to the key. Both the key and the value can be of any type.

The dictionary holder also has a property that keys of class String entered in the Properties dialog automatically generate events and messages for each initial key. For example, the keys Dave, Mary, and Sam would generate the automatic element events: **changedDave**, **changedMary**, and **changedSam**. They would also generate the element messages **Dave**, **setDave**; **Mary**, **setMary**; **Sam**, and **setSam**.

There are a number of other messages for manipulating and processing Dictionaries. Some of these are documented in the Messages section later in this chapter.

**Events**

These events are available in addition to the events described in the *Value Holder* chapter later in this document.

- **changedkey: aValue**
  Occurs whenever the value corresponding to **key** is changed.
  **Triggered By:**
  - **setkey:**
  - **at:put:**

- **keyException: aKey**
  Occurs when it receives a message accessing a nonexistent key.
  **Triggered By:**
  - **at:**
  - **at:put:**
  - **removeKey**
  - **setKey:**
Messages

These messages are available in addition to the messages described in the Value Holder chapter later in this document.

asArray
Translates the values in the Dictionary into an Array. You can place the result into an array holder by using a result link to the part’s setValue: message.

Returns: an Array
Triggers: —

asOrderedCollection
Translates the values in the Dictionary into an OrderedCollection. You can place the result into an ordered collection part by using a result link to the part’s setValue: message.

Returns: an Ordered Collection
Triggers: —

asSet
Translates the contents of the dictionary holder into a member of class Set, eliminating any duplicate elements. You can create a part for holding a Set by setting the class of a value holder to Set.

Returns: a Set
Triggers: —

asSortedCollection
Translates the contents of the dictionary holder into a member of class SortedCollection containing the elements of the dictionary holder sorted in ascending order. You can create a part to hold a SortedCollection by setting the class of a value holder to SortedCollection.

Returns: a Sorted Collection
Triggers: —

at: aKey
Returns the value stored at aKey. Triggers keyException: if the dictionary key specified by aKey does not exist.

Returns: a Value
Triggers: keyException:

at: aKey put: aValue
Stores aValue at index aKey. Triggers keyException: if the dictionary key specified by aKey does not exist.

Returns: aValue
Triggers: keyException:

includes: aValue
Returns true if the dictionary contains an element equal to aValue, otherwise returns false.

Returns: a Boolean
Triggers: —
includesKey: aKey
Returns true if the dictionary holder contains the key aKey, otherwise returns false.
Returns: a Boolean
Triggers: —

isEmpty
Returns true if the size of the dictionary holder is zero, otherwise it returns false.
Returns: a Boolean
Triggers: —

key
Automatically generated for each key entered in the dictionary holder's property dialog. Returns the value at key.
Returns: a Value
Triggers: —

keyAtValue: aValue
Returns the key in the dictionary holder whose paired value equals aValue. Triggers valueException: if the value is not found.
Returns: a Key
Triggers: valueException:

keys
Returns a Set of all the keys in the dictionary which can be displayed in a list pane. You can create a part to hold a Set by giving a value holder the class Set.
Returns: a Set Of Keys
Triggers: —

keysSorted
Returns a member of class SortedCollection containing all of the keys in the dictionary holder. You can create a part to hold a SortedCollection by setting a value holder's class to SortedCollection.
Returns: a Sorted Collection Of Keys
Triggers: —

occurrencesOf: aValue
Returns the number of values in the dictionary holder that are equal to aValue. Note that aValue is not a key.
Returns: an Integer
Triggers: —

removeAll
Deletes the entire contents of the dictionary holder.
Returns: —
Triggers: —
Dictionary Holder

**removeKey: aKey**
Returns the contents of the dictionary holder with the key/value pair whose key is aKey removed. Triggers keyException: if the key is not found.

*Returns:* a Dictionary
*Triggers:* keyException:

**setkey: aValue**
Automatically generated for each key entered in the dictionary holder's property dialog. Sets the value for key to aValue.

*Returns:* aValue
*Triggers:* changedkey:

**size**
Returns the number of elements in the dictionary holder.

*Returns:* an Integer
*Triggers:* —

**updateValue: aDictionary**
Updates the values in the dictionary with the values stored at the same keys in aDictionary.

*Returns:* aDictionary
*Triggers:* valueException:

**Direct-Edit**
Direct-editing the dictionary holder part changes the part name that appears beneath the icon in the workbench.

It is a good idea to change the part name to indicate the purpose of the dictionary holder, just as you would name a variable or a constant in a programming language.

**Property Edit**
The dictionary holder has the same properties as the value holder, except that you cannot change the class of the dictionary holder. In addition to these properties, you can enter initial values into the dictionary.
The part-specific properties of the dictionary holder are explained in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

**New value**
This is an entry field that lets you type an expression for a value, in the same format that would be used in a script.

**Edit value...**
Press this button to display a dialog that lets you enter and edit the initial key/value pairs that are stored in the dictionary. The list pane on the left contains the keys that have been defined for this dictionary. All values entered in this dialog are treated as Strings. If you want to include values other than Strings, you should use the New value entry field.

- **Key:**
  Enter a String for a new key value in this field.

- **Value:**
  Enter the value which corresponds to the key in this field.

- **Add**
  Press the Add button to add a new key/value pair to the initial dictionary contents.
Dictionary Holder

- **Remove**
  Press the Remove button to delete the key/value pair whose key is selected in the list pane.

- **Remove All**
  Remove all initial key/value pairs from the Dictionary.

**To add a new key/value pair to the initial contents of a dictionary holder:**

1. Enter the new key in the Key field.
2. Enter the new value in the Value field.
3. Press the Add button.

The new key appears in the list pane.

Repeat steps 1 through 3 for each additional key/value pair to be added to the initial contents of the dictionary holder.

**To change a value in the dictionary holder:**

1. Select the key in the list pane.
2. Enter the new value in the Value field.
3. Press the Add button.

The new value will be recorded.

**To delete a key/value pair from the initial contents of the dictionary holder:**

1. Select the key to be removed in the list pane.
2. Press the Remove button.

The key will disappear from the list pane.

**Support element messages and events**

If checked, two messages (*key* and *setkey:* ) and one event (*changedkey*) are automatically generated for each initial key in the dictionary holder. These are listed in the Events and Messages tables. The default is to automatically generate element events and messages.

**Related Parts**

- **Value Holder**
  The dictionary holder is similar to a value holder with the type permanently set to Dictionary, except that *key* messages and events can be automatically generated.

**Remarks**

Key access events and messages are only generated for string keys. Any type of key can be used when the *at: and atput:* messages are used.

The hint shown in the information area for a dictionary holder shows all of the values the dictionary holder currently contains. The values are also listed in the Properties dialog via the Edit value... button.
A disk accessor is a nonvisual part that provides access to files and directories on the disk.

The Disk Accessor maintains a current drive, directory path, and a filter pattern for selecting files ("*.txt" for example).

In addition to specifying the current directory and pattern, other operations include:

- Creating and deleting directories
- Testing for the existence of a directory
- Accessing the drive, path, or name of a directory
- Obtaining a list of files and subdirectories within a directory

**Events**

directoryException: aString
   Occurs when a directory operation is attempted and fails. The pathname is provided in aString.

Triggered By: all disk accessor messages that require a directoryString as an argument

drive: driveString
   Occurs when setDirectory: changes the current drive. The value driveString contains the new drive letter followed by a colon.

Triggered By: setDirectory:

existsTrue: directoryString
   Occurs if the directory named by the string passed to the exists: message actually exists. The event value is the pathname of the directory.

Triggered By: exists:

files: anArrayOfStrings
   Occurs whenever the current list of files changes. The names of files in the current directory are provided by the event value.

Triggered By: setDirectory:
setPattern:
operationException: aString
   Occurs when any error occurs during a directory operation.
   Triggered By:  create:
   delete:
   files
   filesIn:
   filesIn:matching:
   filesMatching:
   knownDrives
   subdirectories
   subdirectoriesIn:

pathName: aString
   Occurs whenever the pathname changes. The new pathname is provided in
   aString.
   Triggered By:  setDirectory:

subdirectories: anArrayOfStrings
   Occurs whenever the list of subdirectories changes. The names of the
   subdirectories in the current directory are provided by the event value.
   Triggered By:  setDirectory

Messages

create: directoryString
   Creates the directory whose absolute or relative path is given in
   directoryString. If the new directory is a subdirectory of the current directory,
   the subdirectories: event is triggered. Returns true if the directory is
   successfully created. Returns false otherwise and triggers the
   directoryException: event.
   Returns: a Boolean
   Triggers: directoryException:
   operationException:
   subDirectories

delete: directoryString
   Deletes the directory whose absolute or relative path is given in
   directoryString. If the deleted directory was a subdirectory of the current
   directory, the subdirectories: event is triggered. Returns true if the directory
   is successfully deleted, otherwise returns false and triggers the
   directoryException: event.
   Returns: a Boolean
   Triggers: directoryException:
   operationException:
   subDirectories

directoryName: directoryString
   Returns a string with the unqualified name for the directory indicated by
   directoryString.
   Returns: a String
   Triggers: directoryException:
directoryPath: directoryString
Returns the full pathname of the specified directory.
Returns: a String
Triggers: directoryException:

driveName: directoryString
Returns a String with the drive letter for the specified directory followed by a colon.
Returns: a String
Triggers: directoryException:

exists: directoryString
Triggers the existsTrue: event and returns true if directoryName exists.
Returns false and triggers the directoryException: event if it doesn’t.
Returns: a Boolean
Triggers: existsTrue:
directoryException:

files
Returns an Array of Strings, sorted alphabetically, which are the unqualified
names of the files contained in the current directory which match the current
pattern string.
Returns: an Array of Strings
Triggers: operationException:

filesIn: directoryString
Returns an Array of Strings consisting of the filenames in the specified
directory that match the current pattern.
Returns: an Array of Strings
Triggers: directoryException:
operationException:

filesIn: directoryString matching: aString
Returns an Array of Strings consisting of the filenames in the specified
directory that match the pattern string specified by the second argument.
Returns: an Array of Strings
Triggers: directoryException:
operationException:

filesMatching: aString
Returns an Array of Strings consisting of the filenames in the current directory
that match the specified pattern string.
Returns: an Array of Strings
Triggers: operationException:

knownDrives
Returns an Array of Strings with the known drives.
Returns: an Array of Strings
Triggers: operationException:
Disk Accessor

**pathName**
Returns the fully qualified pathname of the current directory.

*Returns:* a String

*Triggers:* —

**pathNameOfFile: fileName**
Returns the fully qualified pathname of the specified filename in the current directory without checking to see if the file currently exists.

*Returns:* a String

*Triggers:* —

**pathNameOfFile: fileString in: directoryString**
Returns the fully qualified pathname of the specified filename relative to the specified directory. Does not check to see if the file exists.

*Returns:* a String

*Triggers:* directoryException:

**pattern**
Returns the pattern string used to filter the filename list for the current directory.

*Returns:* a String

*Triggers:* —

**setDirectory: directoryString**
Sets the current directory to be the argument, a full or relative pathname, and returns the argument `directoryString`.

*Returns:* directoryString

*Triggers:* directoryException:

**setPattern: aString**
Sets the filter string to the argument, triggers the `files:` event, and returns the argument. The filter string contains wildcards that are used to select filenames. An example of a filter string is ".TXT."

*Returns:* aString

*Triggers:* files:

**subdirectories**
Returns an Array of Strings which are the unqualified names of the subdirectories contained in the current directory. Includes initial entry "." if current directory is not the root directory "\".

*Returns:* an Array of Strings

*Triggers:* operationException:

**subdirectoriesIn: directoryString**
Returns an Array of Strings which are the unqualified names of the subdirectories contained in the specified directory.

*Returns:* an Array of Strings

*Triggers:* directoryException:

operationException:
Direct-Edit

Direct-editing the disk accessor part changes the label that appears beneath the icon in the workbench.

Property Edit

The part-specific control in the disk accessor's Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Pattern

The pattern string is used as a filter to select filenames. It can contain the wildcard character "*" that matches any number of characters. The default pattern is "*." which matches all filenames, with or without an extension. Here are other pattern examples:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.txt</td>
<td>filenames with the extension TXT</td>
</tr>
<tr>
<td>d*.par</td>
<td>filenames that begin with d and contain a PAR extension</td>
</tr>
<tr>
<td>abc.*</td>
<td>filenames with the name abc and any extension</td>
</tr>
</tbody>
</table>

Remarks

The `setDirectory:` message sets the current directory. The argument string can be a drive letter, or a partial or fully qualified pathname.

Before setting the current directory with a relative pathname, be sure to set it to an absolute path, but use caution when specifying drive letters and absolute pathnames in an application because those drives and directories may not exist on other systems.
Example

The example allows you to preview the contents of text files stored on disk. When you type a directory name in the Set Directory: entry field and press Enter, the `setDirectory:` message is sent to the disk accessor part. If the directory name is invalid, the `directoryException:` event is triggered, which causes the Speaker to beep. If it is an existing directory name, the subdirectory and file lists are updated in response to events from the disk accessor.

When the application window opens, the current drive and pathname are displayed. These fields are updated whenever the `pathName:` event fires to indicate that the current directory has changed.

The default pattern is initially displayed in the Set Pattern: entry field. If you type a new pattern in this field and press Enter, the file list changes to display only those files in the current directory that match the pattern.

Type a name in the Set Directory: entry field and press the Directory exists? button to see if the directory does exist. Press the Create directory button to create the directory named in the first entry field or Delete directory to delete it.

This example is in the file DISKACC.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The DLL Accessor is a nonvisual part that provides access to procedures that are contained in dynamic link libraries.

Dynamic linking is a way of accessing functions that are not built into your application or that are written in another programming language.

Using the DLL Accessor's Properties dialog, you specify the information required to access the procedures in a dynamic link library. This required information includes the:

- Name of the DLL file that contains the procedures
- Names of the procedures to be accessed
- Calling convention to be used
- Number and types of the procedure arguments
- Type of the procedure result

You can find this information in the programmer's reference for the DLL file being accessed.

You add a message to a DLL accessor for each procedure you need to access in that DLL file. There is one colon in the message name for each argument required by the DLL procedure.

Each DLL accessor in your application provides access to one DLL file. You can use multiple DLL accessor parts if you need to access multiple DLL files.

The DLL file must be opened before any procedures in it are accessed. You can open it with the open message, or it is opened automatically the first time a message is sent to it to access a DLL procedure. If the DLL file can't be opened for some reason, the openException: event is triggered with the Windows or OS/2 error code as the event value.

If a call fails for any reason, such as there isn't any such procedure in the DLL file or the wrong types of arguments are used, the callException:arguments: event triggers.

When you are through with a DLL file, it is a good idea to close it with the close message because this frees system resources, including the memory containing the library.
Operation

For each procedure you want to access in a DLL file, you define a corresponding message for the DLL accessor. This message name has one colon for each argument the procedure requires.

Events

**openException**: `anInteger`

Occurs when an attempt to open the DLL file fails. The argument `anInteger` contains the Windows or OS/2 error code.

*Triggered By:* `open`

**callException**: `aString arguments: aCollectionOfValues`

Occurs when a call to a DLL procedure fails for any reason other than the DLL library can't be opened. The message that failed and a description of the failure is provided by `aString`, and the message arguments are contained in the collection.

*Triggered By:* `procedureName`

Messages

**close**

Closes the DLL file.

*Returns:* —

*Triggers:* —

**open**

Opens the DLL file so that procedures in it can be executed.

*Returns:* —

*Triggers:* `openException:`

**procedureName**

Invokes the corresponding DLL procedure that you have defined in the Properties dialog. The DLL file is opened if the `open` message hasn't been executed.

*Returns:* `whatever you define`

*Triggers:* `callException:arguments:`

Direct-Edit

Direct-editing the DLL accessor changes the part name that appears beneath the icon in the workbench.
Property Edit

You define the interface to a DLL file with the Properties dialog. It is used to define the messages which interface to the procedures in a DLL file. Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

**DLL filename**

Enter the filename (without directory path) of the DLL file in this field. The DLL file must be in a directory in your PATH environment variable. Do not specify the DLL file extension.

**Procedures**

Lists the DLL accessor messages that have been defined.

**Add...**

Opens the following dialog to prompt for the name of the new message to add to the DLL accessor:

Enter the name of the message to be added to the DLL accessor to access a specific procedure in a DLL. Use one colon in the message name for each parameter of the DLL procedure.

**Delete**

Removes the selected message from the list of messages.
Edit...

Opens a dialog that allows you to specify the following information for the selected message:

- **Entry point of the corresponding DLL procedure**
- **Calling convention to be used**
- **Type of each procedure argument**
- **Return type**

**Entry point**

Enter the entry point in the DLL file of the procedure to be invoked when the message being defined is executed by the DLL accessor. The documentation for the DLL file you are accessing should contain the entry point names, calling convention, and argument information for each of the procedures in the DLL file.

In OS/2, you must call system functions by number. The file ORDINALS.TXT (located in the DOC subdirectory of your installation directory) contains a list of OS/2 system functions and their corresponding numbers. In the Entry point field, the system function number must be proceeded by a "#" symbol.

**Calling convention**

Select the calling convention used by this entry point. 16-bit C, 16-bit Pascal, 32-bit C, and 32-bit Pascal are supported.

**Arguments**

You must set a type for each of the arguments in this list. The argument names are taken from the message name. The text up to the first colon is the name of the first argument, the text up to the next colon is the name of the second argument, and so on. When you make a connection to this message in the workbench, there is one argument connection for each argument.
• **Argument type**
  Select one of the following types for each argument.

  - **short**: A 16-bit signed integer. Use an Integer argument for the matching message.
  - **ushort**: A 16-bit unsigned integer. Use an Integer argument for the matching message.
  - **long**: A 32-bit signed integer. Use an Integer argument for the matching message.
  - **ulong**: A 32-bit unsigned integer. Use an Integer argument for the matching message.
  - **boolean**: An 8-bit value. Use a Boolean argument for the matching message.
  - **handle**: A Windows resource handle. You must use a script to obtain a handle from one of the pane parts (window, group pane, etc.) for the corresponding message argument.
  - **reference**: A pointer to a C structure. The C Structure value message provides the reference argument for the corresponding message.

  DLL calls that require a string parameter and return a reference to this string, must use a C structure part to represent the string parameter. If this parameter is supplied by any other means (for example, a string holder), nil will be returned as the result. If the DLL allocates its own memory for the return value, this limitation does not apply.

  In this case, a reference type procedure argument can be set to a String value. This is similar to the reference type field in a C Structure, which can also be set to either a C Structure or a String value. When the DLL procedure call is made, a reference argument with a String value is passed to the procedure as a pointer to the characters in the String with a null byte appended. The terminating null byte is not included in the result.

  In all cases, if the string parameter is modified by the DLL, the new value will be reflected in the original string which was passed into the DLL accessor.

  To use a returned pointer to a null terminated string, you need to use a String Holder. Create a result link from the DLL accessor call message to the String Holder message setFromReference:.

  See the C Structure chapter earlier in this document for more information about using reference types.

  - **double float**: An 8 byte float value. A double float supports IEEE double float.

• **Return type**
  Set the return type of the procedure to whichever of the types above is appropriate. Select none for a C procedure with either a void return type or a Pascal procedure.

**Related Parts**

• **C Structure**
  A C structure part defines the layout of a record that can be passed or accessed as a DLL procedure reference argument.
Remarks

The DLL file being accessed must reside in a directory listed in the PATH variable.

A special situation exists when a DLL call uses a reference to a string as an argument, and returns this same reference. In such a case, you must use a C structure to represent the string (instead of a string holder, for example). If the DLL call allocates its own memory for the return value, this limitation does not apply.

Example for Windows

This application demonstrates calling a DLL by calling a system function. The Test menu contains commands to enable and disable the list pane. When either command is invoked, the list pane is enabled or disabled and the IsWindowEnabled system function is called. The list pane window handle is passed as an argument and the Boolean return value is displayed in the bottom entry field. Notice that the system function is also called when the window is first opened, so that the enablement status is always displayed correctly.

The test menu is also a popup menu. Right clicking anywhere in the application window will bring up this menu at the current cursor location.

This example is in the file DLEXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Example for OS/2

This application demonstrates calling a DLL by calling a system function, DOSBEEP. In OS/2, you must call system functions by number. The number corresponding to DOSBEEP is 286. Numbers for other OS/2 system functions are listed in ORDINALS.TXT located in the DOC subdirectory of your installation directory.

The example is in the file DLEXMPO.PAR, in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Drawn Button

A drawn button is a visual part that is operated by clicking on it or by pressing the Enter key when the button has the input focus.

The drawn button behaves like a push button except that it has a bitmap for a label. The Windows version of this part also cannot be set as a default button.

The illustration below shows an example of how drawn buttons might be used to create a help browser for various parts:

When a drawn button is resized, the bitmap is automatically scaled to fit the new size of the button.

Operation

To operate a drawn button, click on it. The border of the drawn button changes to indicate that the button is depressed. The clicked event is triggered when you release mouse button 1.

Events

clicked
- Occurs when the user clicks with mouse button 1 or the Enter key is pressed when the button has the input focus.
  
  Triggered By: the user

rightClicked
- Occurs when the user clicks with mouse button 2.
  
  Triggered By: the user
Drawn Button

**tabbed**

Occurs when the Tab key is pressed and the button has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

*Triggered By: the user*

**Messages**

**bitmap**

Returns the button’s bitmap.

*Returns: a Bitmap*

*Triggers: —*

**disable**

Deactivates the button so that it does not respond to user input. A disabled button beeps when you try to press it.

*Returns: —*

*Triggers: —*

**enable**

Enables the button so it responds to user input.

*Returns: —*

*Triggers: —*

**setBitmap: aBitmap**

Sets the button’s bitmap to `aBitmap`.

*Returns: aBitmap*

*Triggers: —*

**setFocus**

Moves the input focus to the drawn button. Input focus is indicated by a dashed line around the button.

*Returns: —*

*Triggers: —*

**Direct-Edit**

Direct-editing a drawn button opens the Bit Editor so you can either draw a new button bitmap or import a bitmap from a file. See the Edit bitmap property in *Appendix C* for information on using the Bit Editor.
Property Edit

The part-specific controls in the drawn button’s Properties dialog are described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

![DrawnButton Properties](image)

Edit graphic...

Opens the Bit Editor so you can either draw a new button bitmap or import a bitmap from a file.

Styles

- **Default button**
  This property, which is only available in the OS/2 version of this part, applies to buttons in dialog windows only. When the focus is not on a push button, a drawn button, or text pane and you press the Enter key, the default push button or drawn button (if there is one) will be activated as if it were pressed. If the focus is on any push button or drawn button, it automatically becomes the default button even though it does not have the Default button style. You can use the arrow keys to get to such a button, and you will notice that the highlighting of the button moves from the previous default button to this new one. Now, if you press either the Space bar or the Enter key, you will select the push button or drawn button.

Related Parts

- **Push Button**
  The push button part has text for a label, rather than a bitmap graphic.

Example

See the example in the Push Button chapter later in this document.
Drop-Down List

A drop-down list is a visual part that works just like a list pane except that the item list is displayed only when a selection is being made.

Only one item can be selected from a drop-down list. The selected item is always displayed. The list of selections appears when the button to the right of the drop-down list is pressed, and disappears when an item is selected or the Escape key is pressed to cancel the selection.

Operation

A new selection triggers the changed: and changedIndex: events.

You can set the items in the drop-down item list to an Array of Strings with setList:, or you can obtain the list of items with the list message.

The currently selected item is accessed with value, and the selection is changed by position with setValueIndex: or by content with setValue:

The following illustration shows a dialog containing two drop-down lists, with the right one open to make a selection.

To make a selection in a drop-down list:

1. Click on the downward pointing arrow.
2. Make a selection from the displayed list by clicking on the desired item.

If the selection list is displayed with a scroll bar, you can scroll the selection list by dragging the scroll box, by clicking on either of the arrows, or by using the up and down arrow keys on your keyboard.

NOTE: To click on a drop-down list in the PARTS Workbench for selecting or editing, you must position the cursor over the gray button or the visible part of the list which shows the current selection.
Drop-Down List

Events

aboutToChange
Occurs when an item is selected. If the abortChange message is executed as a consequence of this event, the selection is canceled and the changed: event is not triggered.

Triggered By: the user

changed: aString
Occurs when the selection changes. The event value is the newly selected item.

Triggered By: the user

triggered:

setValue:

setValueIndex:

changedIndex: anInteger
Occurs when the selection changes. The event value is the index of the newly selected item.

Triggered By: the user

setValue:

setValueIndex:

rightClicked
Occurs when the user clicks on the drop-down list with mouse button 2.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the drop-down list has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

Messages

abortChange
Cancels the selection if executed as a consequence of triggering the aboutToChange event.

Returns: —
Triggers: —

deleteAll
Removes all items from the selection list.

Returns: —
Triggers: —

deleteItem: aString
Removes the item from the selection list that matches aString.

Returns: —
Triggers: —
disable
Ignores user input. The down arrow is dimmed when the drop-down list is disabled.

Returns: —
Triggers: —

enable
Enables the drop-down list to respond to user selections.

Returns: —
Triggers: —

insertItem: aString
Appends aString to the end of the selection list.

Returns: —
Triggers: —

list
Returns the items in the selection list.

Returns: an Array of Strings
Triggers: —

setFocus
Moves the input focus to the drop-down list.

Returns: —
Triggers: —

setList: aString or anArrayOfStrings
Sets the items in the selection list to the items in the argument and returns the new list.

Returns: an Array of Strings
Triggers: —

setValue: aString
Sets the current selection to the first item in the selection list that contains the same text as the argument. If no item in the selection list matches the argument, the selection is not changed and no events are triggered.

Returns: aString
Triggers: changed:
changedIndex:

setValueIndex: anInteger
Sets the current selection to the item whose ordinal position in the selection list (starting at one) is the argument. If the value of the argument is not between one and the number of items in the list, the selection is not changed and no events are triggered.

Returns: anInteger
Triggers: changed:
changeIndex:
Drop-Down List

value
Returns the currently selected item, nil if the list is empty or nothing is selected.

Returns: a String
Triggers: —

valueIndex
Returns the index (starting with one) of the currently selected item, nil if the list is empty or nothing is selected.

Returns: an Integer
Triggers: —

Direct-Edit
Direct-edit lets you to enter the list of items into the drop-down list. Use the Enter key to separate items.

Property Edit
The part-specific controls in the drop-down list’s Properties dialog are described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Edit contents...
Opens a text pane so you can enter or edit the item list. Each line in the text pane becomes an entry in the selection list.
Drag drop...

Opens the following dialog where you can enable the capability for objects to be dragged from and/or dropped on the part:

The Source and the Target check boxes in the Drag group pane are used to enable (or disable) the part as the source and/or the target for a specified set of drag and drop operations.

The Unused and the Chosen and Prioritized list panes in the Drag Target Formats group pane are used to indicate how the part should render a dropped object, and they are disabled unless the part has been enabled as a target. This target part attempts to render the dropped object in one of the chosen formats, beginning with the first entry in the Chosen and Prioritized list pane. If the object cannot be rendered in that format, then the other chosen formats are tried. If the object cannot be rendered in any of the chosen formats, then it is ignored. To move formats between the Unused and the Chosen and Prioritized list panes, drag them from one list to the other. To change the priority order of the chosen formats, drag them within the Chosen and Prioritized list pane.

The Select Operations multiple-selection list pane in the Drag Target Operations group pane is used to indicate the set of drag and drop operations that the part will accept, and it is disabled unless the part has been enabled as a target. An operation is selected if it is highlighted.

The Defaults button is used to restore the part's drag drop property to its default values.

Related Parts

- **Combo Box**  
  A combo box looks like a drop-down list, but is really a form of entry field because arbitrary text can be entered into the text field.

- **Menu**  
  A menu is a list of items that appears on demand. One item can be selected.
• **List Pane**
  
  A list pane is like a drop-down list except that the item list is visible at all times.

**Remarks**

The size of the drop-down list, as shown by the selection handles, controls how much of the selection list is displayed when the button to the right of the selected item is pressed.

**Example**

This application demonstrates selecting items from a drop-down list. When you click on the item in the drop-down list (located in the upper left corner of the window), the **changed** and **changedIndex** events are triggered. The event values are displayed in static text parts to the right of the event labels. If you enter an item in the setValue: entry field, that value is selected if it is in the list. Similarly, changing the setValueIndex: entry field changes the selection by index. You can append items to the list by entering them into the insertItem: entry field.

When you select an item in the drop-down list, the **aboutToChange** event is triggered. This event is linked to LinkJunction1’s **triggerIfFalse** message. If the Allow changes? check box is not checked, the link junction sends the **abortChange** message to the drop-down list and the selection does not change. You can, however, still change the selection in the drop-down list by using the setValue: and setValueIndex: entry fields. If the Allow changes? check box is checked, then the link junction’s **triggered** event is not triggered and the change is allowed.

This example is in the file DROPDOWN.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Edit Menu

The edit menu is a prefabricated menu for editing operations.

The edit menu is made up of a menu part, some menu text items, and some menu separators. It is no different than a menu containing similar items that you would construct yourself. You must connect a link to the edit menu’s `popUp` message, such as from a menu label in a window’s menu bar. You must connect the events of the menu text items in the edit menu to other messages in your application. You can easily change its menu text items in the workbench to meet any special requirements.

At runtime it appears as shown in the following illustration:

Events

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.

Messages

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.

Direct-Edit

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.

Property Edit

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.
Related Parts

The edit menu is made up of the following parts. Refer to their chapters later in this document for information about the events and messages to use with an edit menu.

- **Menu**
  The menu part responds to the `popUp` message and serves as a pane to contain menu text items and menu separators.

- **MenuItem Separator**
  A menu separator draws a horizontal line between two menu items to separate groups of menu items.

- **Menu Text Item**
  A menu text item triggers the `clicked` event when it is chosen. You should connect this event for each item you want to implement.

Example

See the example in the *Menu* chapter later in this document.
**Entry Field**

An entry field is a visual part that lets you enter, edit, and display single lines of text. When an entry field has focus, pressing the Enter key triggers its **entered** event. The **changed** event is triggered when the contents of the entry field are changed between the time the input focus enters the field and the time the input focus is moved from the entry field by either clicking on another part or pressing the Tab key.

When you move the mouse cursor over an entry field, the cursor changes to an I-beam. When you click on the entry field to move the input focus to it, the text cursor appears. The text cursor indicates the point at which characters will be inserted or overwitten, or a selection will begin.

The following illustration shows the prompter part. It uses an entry field to obtain a line of input from the user.

![Illustration of Entry Field](image)

**Operation**

**To enter text into an entry field:**

1. Click on the text in the entry field.
   
   The text cursor appears at the insertion point.

2. Type the characters to be entered.
   
   Normally, characters typed are inserted. You can replace existing characters by pressing the Insert key before typing. To change from replacing characters back to inserting characters, press the Insert key again.

**To select text so that it can be deleted or copied:**

1. Move the cursor over the entry field to the position where you want to begin the selection.

2. Select the entry field by clicking on it. The text cursor appears.

3. Drag either left or right to make the selection.
To delete text:
> Select the text to be deleted and press the Delete key.

To delete individual characters:
1 Position the text cursor in the entry field.
2 Press the Delete key to delete the character to the right of the text cursor, or press the Backspace key to delete characters to the left of the text cursor.

To copy text to the clipboard:
1 Select the text to be copied.
2 In Windows, press <Ctrl> C. (To cut text instead of copying it, press <Ctrl> X.)
   In OS/2, press <Ctrl> Insert. (To cut text instead of copying it, press <Shift> Delete.)

To paste text from the clipboard:
1 Position the text cursor at the insertion point.
2 In Windows, press <Ctrl> V.
   In OS/2, press <Shift> Insert.

To scroll text:
If the text in the entry field is too long to be displayed, you can scroll it left or right in several different ways:
> Drag the mouse across the entry field opposite of direction that you want the text to scroll.
   or
> Press the Home or End keys to scroll to the beginning or end of the line.
   or
> Use the left and right arrow keys.

To move the text cursor to another position in the entry field:
> Click at the desired position.
   or
> Use the Home or End keys to move the text cursor to the beginning or end of the text.
   or
> Use the left and right arrow keys to move the text cursor to the new position.
Events

`aboutToChangeTo: aValue`
Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. This event is triggered only after `aValue` has been validated by the entry field's built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the `abortChange` and `retryChange` messages to respond to the proposed new value. If this event is triggered by the `clear` or `setValue:` messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its `changed:` event with `aValue` as its event value.

`Triggered By: the user`
`clear`
`setValue:`

`changed: aString`
Occurs when the entry field loses the input focus and the text has changed since the input focus entered the field, or when the `clear` or `setValue:` messages are executed. The new contents are provided in `aString`.

`Triggered By: the user`
`clear`
`setValue:`

`entered: aString`
Occurs when the Enter key is pressed and the entry field has focus. The entry field contents are provided in `aString`.

`Triggered By: the user`

`rightClicked`
Occurs when the user clicks button 2 in the entry field.

`Triggered By: the user`

`tabbed`
Occurs when the Tab key is pressed and the entry field has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

`Triggered By: the user`

`valueException: aValue`
Occurs when the `setValue:` message is sent and the argument is too long (based on the property “Maximum number of characters”) to fit in the entry field.

`Triggered By: setValue:`
Entry Field

Messages

abortChange
Rejects the new value entered by the user and resets the entry field's contents to the original value. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —

clear
Sets the contents of the entry field to an empty string.

Returns: —
Triggers: changed:

disable
Causes the entry field to ignore user input. The text is dimmed in a disabled entry field.

Returns: —
Triggers: —

enable
Enables the entry field to respond to user input.

Returns: —
Triggers: —

retryChange
Rejects the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —

selectAll
Selects all the text in the entry field, when followed by the setFocus message.

Returns: —
Triggers: —

setFocus
Moves the input focus to the entry field. The field has the input focus when the blinking text cursor is displayed.

Returns: —
Triggers: —

setValue: aString
Sets the contents of the entry field to aString.

Returns: aString
Triggers: changed:
valueException:
value

Returns the contents of the entry field.

*Returns:* a String

*Triggers:* —

**Direct-Edit**

Direct-editing the entry field lets you enter the initial contents.

**Property Edit**

Each of the part-specific controls in the entry field’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

Note that the Windows version of this part has the additional property of Convert characters to, and the OS/2 version of this part has the additional property of Alignment.

**Contents**

The initial contents of the entry field can be changed by either direct-editing the entry field or using this field in the Properties dialog.

**Maximum number of characters**

This is the maximum number of characters that can be entered in the field.

**Value when empty**

This is the value returned by the entry field when nothing has been entered. The default is an empty string.
Entry Field

Styles

- **Border**
  When checked, a thin solid border outlines the entry field. The default is to display the border.

- **Auto scroll**
  When checked, the characters in the entry field can be scrolled left and right if they are not entirely visible within the entry field. If this is not checked, the characters cannot be scrolled. You can still enter text that is longer than the entry field, but you cannot scroll it. The default is to allow scrolling.

- **Read only**
  When checked, the contents cannot be changed by typing. The default is to allow changes.

- **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

- **Convert characters to**
  For the Windows version of this part, this property causes the entry field to display characters as all upper case, all lower case, or as they were entered.

- **Alignment**
  For the OS/2 version of this part, the alignment of the text within the entry field can be set to left-aligned, centered, or right-aligned. The default is left-aligned.

Related Parts

- **Combo Box**
  A combo box is an entry field that can have its contents set from a drop-down list of suggestions as well as by typing.

- **Text Pane**
  A text pane is similar to an entry field, but allows the entry and display of multiple lines of text.

Remarks

Pressing the Tab key triggers the **tabbed** event and does not enter a tab character into the entry field. The **tabbed** event causes the entry field to lose focus, which causes the **changed:** event to be triggered if the entry field contents have been changed.

You can enter a tab character in an entry field by using the **setValue:** message or by pasting it from the clipboard.

The text selected by the **selectAll** message is not highlighted until the input focus is moved to the entry field (with the **setFocus** message).
Example

To experiment with an entry field, enter text into the entry field in the upper left corner. The changed: and entered: events are connected to the similarly labeled entry fields. The buttons in the top right corner send various messages to the entry field. Along the bottom of the window are a set of entry fields, each having a different property set. You can experiment with entering text into these fields to see the effects of the different properties.

You would expect pressing the selectAll button to select the characters in the entry field. That’s what it does if you try it out, but if you examine the links from the selectAll button you will find that in addition to the expected link between the button clicked event and the entry field selectAll message, there is also a result link from the selectAll message to the entry field setFocus message. The reason for this is that pressing the selectAll button causes the input focus to be moved from the entry field to the button, and text selections are shown only when the entry field has the input focus. Therefore without the result link, pressing the selectAll button would only have the effect of triggering the changed: event because the input focus was removed from the entry field — the selection in the entry field would not be shown.

The Windows version of this example is in the file ENTRXMPW.PAR, and the OS/2 version is in the file ENTRXMPO.PAR, in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A file accessor is a nonvisual part that provides operations on a file. The file accessor allows you to perform such file operations as the following:

- Reading and writing the contents of a file
- Copying the contents of a file to another file
- Renaming a file
- Deleting a file
- Getting status information (such as size, date, or time) about a file

Each message requires an argument which is a string containing the full pathname of the file. Each event provides the filename as the event value.

When you read the contents of a file, you can set a threshold for the maximum number of bytes to read in the Properties dialog. The `fileTextToThreshold:` message will read the file contents up to this threshold. Setting the threshold to `nil` (the default) means that the entire contents are always read.

**Events**

**existsTrue:** `fileString`

Occurs when the `exists:` message is executed for a file which is present. `fileString` contains the name of the file.

*Triggered By:* `exists:`

**fileException:** `aString`

Triggered when a file operation is attempted and fails. `aString` is a string describing the error.

*Triggered By:* all file accessor messages

**fileNameException:** `fileString`

Occurs when the file named by `fileString` does not exist and cannot be created. For example, when the path name specifies a file in a nonexistent directory.

*Triggered By:* `setFileText:`

- `rename:`
- `copy:`
- `fileName:`
noSuchFileException: fileString
Occurs when the file corresponding to the filename specified in the message argument cannot be located. The fileString contains the name of the absent file.

Triggered By:
copy:to:
date:
dateTimeString:
delete:
descriptionString:
fileText:
fileTextToThreshold:ename:to:
setFileText:to:
size:
time:

sizeException: fileString
Occurs when fileString did not return the entire contents of a file because the file size was greater than the threshold set. The fileString contains the name of the file.

Triggered By:
fileTextToThreshold:

Messages

copy: fileString1 to: fileString2
Copies the file named by fileString1 to the file named by fileString2.

Returns:
—
Triggers:
fileException:
fileNameException:
noSuchFileException:

date: fileString
Returns the date the file named by fileString was last modified.

Returns:
a Date
Triggers:
fileException:
noSuchFileException:

dateTimeString: fileString
Returns a String containing the date and time the file named by fileString was last modified.

Returns:
a String
Triggers:
fileException:
noSuchFileException:

delete: fileString
Deletes the file named by fileString.

Returns:
—
Triggers:
fileException:
noSuchFileException:
**descriptionString: fileString**

Returns a String containing a full description of the file named by `fileString`, including the name, size, and date and time last written.

- **Returns:** a String
- **Triggers:** `fileException`, `noSuchFileException`

**exists: fileString**

Triggers `existsTrue` and returns true if the file named by `fileString` exists.

- **Returns:** a Boolean
- **Triggers:** `fileException`, `existsTrue`

**fileExtension: fileString**

Returns the extension of the name in `fileString`.

- **Returns:** a String
- **Triggers:** —

**fileName: fileString**

Returns the unqualified name of the file named by `fileString`. The unqualified name has the path removed.

- **Returns:** a String
- **Triggers:** `fileException`, `fileNameException`

**fileText: fileString**

Returns the entire contents of the file named by `fileString`, regardless of the threshold setting.

- **Returns:** a String
- **Triggers:** `fileException`, `noSuchFileException`

**fileTextToThreshold: fileString**

Returns the contents of the file named by `fileString`. If the threshold is set and the size of the file exceeds the threshold, then only the characters up to the threshold are returned, prefixed with a message indicating that the file was only partially read, and the `sizeException` event is triggered.

- **Returns:** a String
- **Triggers:** `fileException`, `noSuchFileException`, `sizeException`

**rename: fileString1 to: fileString2**

 Renames the file named by `fileString1` to `fileString2`.

- **Returns:** —
- **Triggers:** `fileException`, `fileNameException`, `noSuchFileException`
**setFileText: fileString to: textString**

Sets the contents of the file named by `fileString` to `textString`. If the file already exists, its contents are replaced with `textString`.

*Returns:* —

*Triggers:* `fileException:`

`fileNameException:`

`noSuchFileException:`

**size: fileString**

Returns the length of the file named by `fileString` in bytes.

*Returns:* an Integer

*Triggers:* `fileException:`

`noSuchFileException:`

**time: fileString**

Returns the time the file named by `fileString` was last modified.

*Returns:* a Time

*Triggers:* `fileException:`

`noSuchFileException:`

### Direct-Edit

Direct-editing the file accessor changes the part name that appears beneath the icon in the workbench.

### Property Edit

The part-specific control in the file accessor's Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

#### Partial read threshold

This field sets the number of bytes that will be read from a file by the `fileTextToThreshold:` message. When set to nil (the default), all bytes in the file are read.
Related Parts

- **Disk Accessor**
  The disk accessor provides information about directories and the files they contain.

- **File Dialog**
  The file dialog lets you browse the disk and select files.

Example

This example allows you to view the statistics for a selected file as well as the first 400 characters of text files.

You can select a file in one of two ways:

- Click on the Pick file... button and use the file dialog.
- Type a name directly in the entry field and press the Enter (or Return) key.

After you select a file, the **trigger** message is sent to the FileSelected link junction with the file name as the argument. The **trigger** message triggers the FileSelected link junction’s **triggered** event with the file name as its event value. This event fires a link which sends the **exists** message to FileAccessor1. The file name is the argument for the **exists** message.

If the file exists, FileAccessor1’s **existsTrue** event is triggered and fires a link that sends the **trigger** message to the FileExists link junction. The event value of **existsTrue** (the file name) is provided as the argument to the **trigger** message.
The **triggered:** event of the link junction is then used to fire five separate links
(one of which is shown in the previous illustration) to the following file accessor
messages:

- `fileName:`
- `fileExtension:`
- `date:`
- `time:`
- `size:`
- `fileTextToThreshold:`

The event value of the **triggered:** event (the file name) is supplied as the argument
to each of these messages. Result links from each message are used to set the values
of the corresponding static text fields.

The result of the **exists:** message (a Boolean) is always sent as an argument to the
**triggerIfFalse:** message of the FileDoesNotExist link junction. If the result is false
(the file does not exist), the link junction triggers its **triggered:** message and fires a
link that sends the **openText:** message to the info dialog. An argument link is used
to retrieve the contents of the entry field for use as the argument to the **openText:**
message. The info dialog opens, informing the user that the file does not exist.

After a file’s statistics are displayed, you can click on the Contents button to display
the first 400 characters of the file in the text pane. The 400 character limit is specified
in the file accessor’s Properties dialog.

This example is in the file FILEACC.PAR in the SAMPLE\PARTDEMO subdirectory of
your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A file dialog is a predefined dialog that lets you browse the file system and select a file. It is a visual part.

There are four messages that open a file dialog. Two of them, **pathNameForOpen** and **pathNameForOpen:** activate file dialogs that have an Open button. The other two messages, **pathNameForSave** and **pathNameForSave:** activate file dialogs that have a Save button. The argument for the messages requiring arguments is a default pattern to filter file names. This default pattern appears in the topmost entry field. The messages without arguments use the pattern set in the Properties dialog for the default pattern.

When you double-click on a filename in the list pane or select a file in the list pane and press the OK button, the **pathName:** event is triggered with the full pathname of the file as the event value. If you press the Cancel button, the **cancel** event is triggered instead.

The files displayed in the dialog can be filtered by a pattern which can either be set in the Properties dialog or passed as an argument to the **pathNameForOpen**, **pathNameForSave**, or **setPattern** messages.

### Events

**cancel**

Occurs when the user presses the Cancel button or closes the dialog using the control menu (in Windows) or the system menu (in OS/2).

*Triggered By:* the user

**pathName:** *fileString*

Occurs when the user chooses a file in the file dialog. The event value *fileString* contains the full path name of the file.

*Triggered By:* the user

### Messages

**pathNameForOpen**

Opens a file dialog for opening a file. Either **pathName:** or **cancel** is triggered depending on whether the user selects a file or presses the Cancel button.

*Returns:* —

*Triggers:* —
**File Dialog**

**pathNameForOpen:** *aString*
Opens a file dialog for opening a file. Either **pathName** or **cancel** is triggered depending on whether the user selects a file or presses the Cancel button. The initial pattern to use for filtering files is *aString*.

- **Returns:** —
- **Triggers:** —

**pathNameForSave**
Opens a Save file dialog. Either **pathName** or **cancel** is triggered depending on whether the user selects a file or presses the Cancel button.

- **Returns:** —
- **Triggers:** —

**pathNameForSave:** *aString*
Opens a Save file dialog. Either **pathName** or **cancel** is triggered depending on whether the user selects a file or presses the Cancel button. The initial pattern to use is *aString*.

- **Returns:** —
- **Triggers:** —

**pattern**
Returns the pattern string.

- **Returns:** *a String*
- **Triggers:** —

**setPattern:** *aString*
Sets the pattern to *aString*.

- **Returns:** —
- **Triggers:** —

**setWaitForAnswer:** *aBoolean*
Sets the file dialog’s Wait for answer property to *aBoolean*.

- **Returns:** *aBoolean*
- **Triggers:** —

**waitForAnswer**
Returns true if the file dialog’s Wait for answer property is on, false otherwise.

- **Returns:** *a Boolean*
- **Triggers:** —

**Direct-Edit**

Direct-editing the file dialog changes the part name that appears beneath the icon in the workbench.
Property Edit

Each of the part-specific controls in the file dialog's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Default pattern

You can enter the initial pattern that determines which file names in a directory should be displayed. The default pattern is "*.*".

Wait for answer

In Windows, the file dialog is a standard dialog and its Wait for answer property is permanently set. See the Dialog Window chapter earlier in this document for information about the Wait for answer property.

Example

This example opens a file dialog when you press the Choose file... button. If you cancel the file dialog, the string Canceled... appears in the entry field. If you select a file, the full filename appears in the entry field.

This example is in the file FILEDIAL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The file menu is a visual part that implements a prefabricated menu for file operations.

The file menu is made up of a menu part, some menu text items, and some menu separators. It is no different than a menu you would construct yourself. To use it, you need to connect a link to the file menu’s `popUp` message (from a window’s menu bar for example) and you need to connect the events of the file menu’s menu text items to other messages in your application. You can easily change the menu text items while in the workbench to meet any special requirements.

At runtime, the file menu appears as shown in the following illustration:

![File Menu](image)

**Events**

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.

**Messages**

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.

**Direct-Edit**

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.

**Property Edit**

See the `Menu`, `Menu Item Separator`, and `Menu Text Item` chapters.
Related Parts

The file menu is made up of the following parts. Refer to their chapters (later in this document) for information about the events and messages to use with a file menu.

- **Menu**
  The menu part responds to the `popUp` message and serves as a pane to contain menu text items and menu separators.

- **Menu Item Separator**
  A menu separator draws a horizontal line between two menu items to separate groups of menu items.

- **Menu Text Item**
  A menu text item triggers the `clicked` event when it is chosen. Connect this event for each item you want to implement.

Example

See the example in the *Menu* chapter later in this document.
Float Entry Field

Float entry fields operate like entry field parts except that they restrict input to Float related characters and format them for display.

The only characters that you may enter in a float entry field are the following:

- digits
- plus and minus signs
- e and E
- decimal point

If you try to enter a different type of character, the system speaker beeps. If you do not enter a valid Float, the following confirmor is displayed:

Clicking on Yes returns the input focus to the field so you can correct your entry. Clicking on No restores the contents to the previous value.

When the float entry field loses focus or you change the contents with the `setValue:` message, the contents are automatically formatted as float.

The float entry field bases its formatting on the operating system number settings. In Windows, you can view and edit these settings via the International icon within the Windows control panel. In OS/2, you can view and edit these settings via the Country object in the System Setup folder.

The value of a float entry field returned by the `value` message and set with the `setValue:` message is a Float. The `valueFormatted` message returns a String that represents the contents as displayed.

The value of an empty float entry field is always nil.

Events

`aboutToChangeTo: aValue`

Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. This event is triggered only after `aValue` has been validated by the entry field's built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.
Float Entry Field

If this event is triggered by the user, you can use the `abortChange` and `retryChange` messages to respond to the proposed new value. If this event is triggered by the `clear` or `setValue:` messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its `changed:` event with `aValue` as its event value.

**Triggered By:** the user

**clear**

**setValue:**

**changed: aFloat**

Occurs when the field loses the input focus and the text has changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. The new contents are provided in `aFloat`.

**Triggered By:** the user

**clear**

**setValue:**

**entered: aFloat**

Occurs when the Enter key is pressed. The contents are provided in `aFloat`.

**Triggered By:** the user

**rightClicked**

Occurs when the user clicks mouse button 2 in the field.

**Triggered By:** the user

**tabbed**

Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

**Triggered By:** the user

**valueException: aValue**

Triggered when the argument of the `setValue:` message is not a valid value for the field.

**Triggered By:** `setValue:`

**Messages**

**abortChange**

Rejects the new value entered by the user and resets the entry field’s contents to the original value. This message can only be sent when the user triggers the `aboutToChange:` event. Its behavior is undefined in other situations.

**Returns:** —

**Triggers:** —

**add: aNumber**

Returns the result of adding `aNumber` to the value in the field.

**Returns:** a Number

**Triggers:** —
clear
Sets the contents to nil.
Returns: —
Triggers: changed:
disable
Ignores user input. The characters in a disabled field are dimmed.
Returns: —
Triggers: —
divide: aNumber
Returns the result of dividing the value in the field by aNumber.
Returns: a Number
Triggers: —
enable
Enables the field to respond to user input.
Returns: —
Triggers: —
multiply: aNumber
Returns the result of multiplying aNumber by the value in the field.
Returns: a Number
Triggers: —
retryChange
Rejects the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.
Returns: —
Triggers: —
selectAll
Selects all the characters in the field.
Returns: —
Triggers: —
setFocus
Moves the input focus to the field. Input focus is indicated by the blinking text cursor.
Returns: —
Triggers: —
setValue: aValue
Sets the contents of the field to aValue. If aValue is not a Float or a String, it is converted to a String before it is validated. If aValue is not a valid currency, a warning dialog is displayed and the contents of the field are not changed.
Returns: —
Triggers: changed:
valueException:
Float Entry Field

subtract: aNumber

Returns the result of subtracting aNumber from the value of the field.

Returns: a Number
Triggers: —

value

Returns the contents of the field (a Float or nil).

Returns: a Float
Triggers: —

valueFormatted

Returns the formatted text string displayed in the field.

Returns: a String
Triggers: —

Direct-Edit

Direct-editing the float entry field changes its value. An Invalid input dialog is displayed if you enter characters that are not allowed in the field.

Property Edit

The part-specific properties of the float entry field are described in the following paragraphs. The OS/2 version of this part has an additional Alignment property.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Contents

You can change the contents of the float entry field by either direct-editing or by typing a value in this field.
Styles

- **Border**
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**
  When checked, the characters in the field can be scrolled left and right if they are not entirely visible within the entry field. If this is not checked, the characters cannot be scrolled. You can continue to enter characters whether or not you can see them. The default is to allow scrolling.

- **Read only**
  When checked, the contents cannot be changed by typing. The default is to allow changes.

- **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

- **Alignment**
  For the OS/2 version of this part, the alignment of the text within the field can be set to left-aligned, centered, or right-aligned. The default is left-aligned.

Related Parts

- **Alphabetic Entry Field**
  Restricts input to letters.

- **Currency Entry Field**
  Restricts input to currency related characters and displays input formatted as currency.

- **Date Entry Field**
  Restricts input to digits and displays input formatted as a date.

- **Entry Field**
  A general entry field part that allows information to be entered in any format.

- **Integer Entry Field**
  Restricts input to integer numbers.

- **Phone Number Entry Field**
  Restricts input to digits and displays input in phone number format.

- **Picture Entry Field**
  Restricts input to a pattern specified in the field’s properties dialog.

Example

FORMAT.PAR, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory, demonstrates all of the formatted entry fields. It is described in the *Alphabetic Entry Field* chapter earlier in this document.
**Format Menu**  
*(Win32 only)*

The format menu is a visual part that implements a prefabricated menu for operations on text using the rich edit part. The format menu is made up of a menu part, some menu text items, and some menu separators. It is no different than a menu you would construct yourself.

In a running application (such as the EDITOR.PAR sample application in the SAMPLE\PARTDEMO directory), the format menu appears as shown in the following illustration:

![Format Menu Illustration](image)

**Operation**

To use this format menu, you need to connect a link to its `popup` message (from a window's menu bar for example) and you need to connect the events of the format menu's menu text items to other messages in your application. You can easily change the menu text items while in the workbench to meet any special requirements.

To enable the user of your application to modify the format of the text in the rich edit part during runtime, link the format menu's menu items to the corresponding messages of the rich edit part:

<table>
<thead>
<tr>
<th>Format Menu Item</th>
<th>Rich Edit Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font</td>
<td>changeFont</td>
</tr>
<tr>
<td>Color</td>
<td>changeColor</td>
</tr>
<tr>
<td>Paragraph</td>
<td>changeParagraph</td>
</tr>
<tr>
<td>Tabs</td>
<td>changeTabs</td>
</tr>
<tr>
<td>Increase size</td>
<td>growTextLimit</td>
</tr>
</tbody>
</table>
Note that the Increase Size command is needed because the system does not automatically increase the capacity of the rich edit control when its limit is exceeded. The initial size limit is 32K characters.

Events

See the Menu, Menu Item Separator, and Menu Text Item chapters.

Messages

See the Menu, Menu Item Separator, and Menu Text Item chapters.

Direct-Edit

See the Menu, Menu Item Separator, and Menu Text Item chapters.

Property Edit

See the Menu, Menu Item Separator, and Menu Text Item chapters.

Related Parts

The format menu is a prefabricated menu for operations on text using the rich edit part. Refer to the Rich Edit (Win32 only) chapter (later in this document) for information about the messages corresponding to the commands on the format menu.

- **Rich Edit (Win32 only)**
  The rich edit part is a control where you can enter and edit text, which can have rich text formatting (RTF) applied at both the paragraph and character level.

The format menu is made up of the following parts. Refer to their chapters (later in this document) for information about the events and messages to use with a format menu.

- **Menu**
  The menu part responds to the popUp message and serves as a pane to contain menu text items and menu separators.

- **Menu Item Separator**
  A menu separator draws a horizontal line between two menu items to separate groups of menu items.

- **Menu Text Item**
  A menu text item triggers the clicked event when it is chosen. Connect this event for each item you want to implement.
Example

The format menu is used in the EDITOR.PAR sample application, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. Commands on the Format menu are linked to operations within the rich edit control (as explained in the Operation topic earlier in this chapter). Some of these commands are also available as toolbar buttons.

For additional explanation of:

- This sample application, see the example in the Toolbar (Win32 only) chapter later in this document
- The rich edit part used in this sample application, see the example in the Rich Edit (Win32 only) chapter later in this document
- Using menus, see the Menu chapter later in this document
Format Menu (Win32 only)
Graph Pane

A graph pane is a visual part that displays bitmaps and icons. The graph pane can also be used as a canvas for drawn graphics.

The graph pane can hold a bitmap or icon larger than the part. Scroll bars are always displayed, even when the image is smaller than the pane. You can use the Stretch bitmap to fit window property to have bitmaps resized (either larger or smaller) to the dimensions of the pane. Note that the vertical and horizontal dimensions are resized independently, so the aspect ratio could be changed.

Events

**changed: aBitmap**

Occurs when the contents of the graph pane change. The event value is the new contents of the graph pane.

*Triggered By:* clearBitmap, pasteFromClipboard, setBitmap:, setFromBitmapFile:, setFromIconFile:

**clicked: aPoint**

Occurs when the user clicks mouse button 1 in the graph pane. The event value is a point representing the hotspot for the cursor, relative to the upper left corner of the graph pane.

*Triggered By:* the user

**fileException: aString**

Occurs when you try to set the bitmap or icon from a file that doesn’t exist or if the file does not contain a valid bitmap or icon. The event value *aString* contains the name of the missing or incorrect file.

*Triggered By:* setFromBitmapFile:, setFromIconFile:

**rightClicked**

Occurs when the user clicks mouse button 2 on the graph pane.

*Triggered By:* the user

**tabbed**

Occurs when the Tab key is pressed and the graph pane has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

*Triggered By:* the user
Graph Pane

Messages

**bitmap**
Returns the portion of the graphic visible in the graph pane.

- **Returns:** a Bitmap
- **Triggers:**

**clearBitmap**
Clears the contents of the graph pane.

- **Returns:** —
- **Triggers:** changed:

**copyToClipboard**
Copies the contents to the system clipboard as a bitmap.

- **Returns:** —
- **Triggers:**

**disable**
Causes the graph pane to ignore user input.

- **Returns:** —
- **Triggers:** —

**enable**
Enables the graph pane to respond to user input.

- **Returns:** —
- **Triggers:** —

**pasteFromClipboard**
Sets the contents from the bitmap in the system clipboard.

- **Returns:** —
- **Triggers:** changed:

**pen**
Returns the pen for the graph pane. Scripts can send messages to the pen to draw on the graph pane.

- **Returns:** a Pen
- **Triggers:** —

**setBitmap:** *aBitmap*
Sets the contents of the graph pane to *aBitmap*.

- **Returns:** *aBitmap*
- **Triggers:** changed:

**setFocus**
Moves the input focus to the graph pane.

- **Returns:** —
- **Triggers:** —
Graph Pane

**setFromBitmapFile: aString**
Sets the contents of the graph pane to the contents of the bitmap (BMP) file whose path is `aString`.

*Returns:* `aString`

*Triggers:* `changed: fileException:`

**setFromIconFile: aString**
Sets the contents of the graph pane to the contents of the icon (ICO) file whose path is in `aString`.

*Returns:* `aString`

*Triggers:* `changed: fileException:`

Direct-Edit

Direct-editing the graph pane opens the Bit Editor so you can revise the bitmap or read it from a file.

Property Edit

The Properties dialog for the graph pane part is shown in the following illustration. Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

![GraphPane Properties Dialog](image)

**Styles**

- **Border**
  If checked, a border is displayed around the pane. The border is displayed by default.

- **Vertical scroll bar**
  If checked, the vertical scroll bar is present. The scroll bar is displayed by default.

- **Horizontal scroll bar**
  If checked, the horizontal scroll bar is present. The scroll bar is displayed by default.
Graph Pane

**Stretch bitmap to fit window**
When this property is checked, the bitmap is resized (either larger or smaller) to fit the dimensions of the graph pane. Note that the vertical and horizontal dimensions are resized independently, so the aspect ratio could be changed.

**Edit graphic...**
Opens the Bit Editor so you can draw the contents or set it from a file. For more information, refer to the Edit bitmap property in Appendix C, Standard Part Properties.

**Related Parts**
- **Static Graphic**
  A static graphic part also displays a bitmap in an application.

**Example**

This example displays a bitmap file in a graph pane. Clicking Pick file... opens a file dialog that lets you select a bitmap file (BMP) for display. The graphic can be then copied to or pasted from the system clipboard by clicking the appropriate buttons. When the user clicks on the graph pane, the **clicked:** event triggers and updates the static text field labeled Clicked: with the point at which the mouse was clicked. This example is in the file GRPHXmpl.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
**Group Pane**

The group pane, a visual part, is a pane (with a label and a line around the border) that is used to visually group other visual parts.

Group panes are used for visual effect only and do not structurally contain other visual parts as children.

The following illustration shows two groups of radio buttons that select paragraph attributes. Each group of radio buttons is in its own group pane.

![Group Pane Illustration](image)

**Events**

The group pane has no events.

**Messages**

**label**

Returns the label.

- **Returns:** a String
- **Triggers:** —

**setLabel: aString**

Sets the label.

- **Returns:** aString
- **Triggers:** —

**Direct-Edit**

Direct-editing a group pane lets you enter the text for the label.
Property Edit

The part-specific control in the group pane’s Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Label

You can set the label that appears on the group pane either in this field or with direct-edit.

Related Parts

- **Radio Button**
  A radio button allows you to choose one alternative from several.
Header Control
(Win32 only)

The header control is a row of buttons that typically is used in conjunction with a list of data items formatted in columns (such as the Report view of a List View part). This part implements the Windows 95 Header control. A header control sends notifications (events) when its columns are clicked or resized by the user.

Here is an example of a header control in a moderately complex application:

![Image of a header control example]

**Operation**

Each button in the header control is typically aligned over a column of data, and the associated header button can be clicked by the user to trigger some action on that column of data (such as sorting by that column). The dividers between header items may be clicked and dragged by the user to adjust the widths of the columns (typically for both the headers and their associated list columns).

**Events**

A header control has the following events.

- **clicked: anObject**
  - The header item specified by the event argument was clicked.
  - Triggered By: the user

- **clickedIndex: anInteger**
  - The column specified by the event argument was clicked.
  - Triggered By: the user

- **dividerDoubleClicked: anObject**
  - The divider to the right of the header item specified by the event argument was double-clicked.
  - Triggered By: the user
Header Control (Win32 only)

**dividerDoubleClickedIndex:** `anInteger`

The divider to the right of the column index specified by the event argument was double-clicked.

*Triggered By: the user*

**resizedIndex:** `anInteger to: anotherInteger`

The divider to the right of the column index specified by `anInteger` was resized to the width in pixels specified by `anotherInteger` (and the mouse button has been released).

*Triggered By: the user*

**resizingIndex:** `anInteger to: anotherInteger`

The divider to the right of the column index specified by `anInteger` is being resized to the width in pixels specified by `anotherInteger` (and the mouse button is still down).

*Triggered By: the user*

**tabbed**

Occurs when the Tab key is pressed and the control has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

*Triggered By: the user*

**Messages**

A header control has the following messages.

**columnWidths**

Answer the current widths (in pixels) of the heading columns.

*Returns:* an Integer

*Triggers:* —

**contents**

Answer the contents of the receiver.

*Returns:* a Collection of HeaderItems

*Triggers:* —

**contents:** `aCollection`

Set the headings to `aCollection` of Strings or HeaderItems.

*Returns:* a Collection

*Triggers:* —

**disable**

Disable user interaction with the header control.

*Returns:* —

*Triggers:* —

**enable**

Enable user interaction with the header control.

*Returns:* —

*Triggers:* —
setFocus

Moves the input focus to the header control.

Returns: —

Triggers: —

Direct-Edit

Direct-editing the header control allows you to enter the strings for the column labels, separated by tabs.

Property Edit

Each of the part-specific controls in the header control’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Buttons

When this property is checked, the column headings are displayed as three-dimensional push buttons. When this property is unchecked, the column headings are displayed flat. By default, the column headings are displayed as three-dimensional push buttons.
Header Control (Win32 only)

Related Parts

- **List View (Win32 only)**
  The list view is a visual part that can display a collection of items several different ways: in icon view, small icon view, list view, or report view.
A help menu is a pre-built menu (similar to the file menu and the edit menu) that provides access to the table of contents, index, and other features of the help system for your operating system. You can use the part as is, or you can modify it.

Each help menu is associated with a specific window. When you choose the Help/Contents command (in Windows) or the Help/General help command (in OS/2), the help system displays the corresponding contents topic from the help file specified in the window’s End-user help file property. If your application uses different windows to display different views of the same information, you should specify the same help file for each window so that you get a consistent result when you request an index or table of contents.

The following illustration shows how the help menu appears in a running application in Windows 95:

![Windows Help Menu Illustration]

The following illustration shows how the help menu appears in a running application in OS/2:

![OS/2 Help Menu Illustration]
Operation

Both Windows and OS/2 support context-sensitive help. An application developed in PARTS Workbench can provide context-sensitive help for visual controls, menus, and menu items that are displayed in a window. You request help for a specific control by pressing the F1 key while the control has focus. You request help for menus and menu items by first selecting the menu or menu item and then pressing the F1 key.

Windows and OS/2 each provide their own compiler for creating help files. Refer to the documentation for your operating system's help compiler for information on creating help files.

Assigning a Help File to a Window

You must tell the help system where to look for help when you request it. You specify a help file during application development with the window part's End-user help file property. When you request help for any child of the window, the help system looks for the requested topic in the specified help file. You can create a separate help file for each window or you can share the same help file among multiple windows.

If each window represents a different view on the same application, the windows should share the same help file so that when you request the index or table of contents for the help file, you always get the same thing regardless of the view.

If each window represents a separate sub-application, you may want to use separate help files for ease of maintenance. If one sub-application changes, you only need to recompile and distribute the relatively small portion of the overall help system that applies to the changed sub-application.

Identifying Help Topics

PARTS Workbench automatically generates strings for each visual control, menu, and menu item. The string for a visual control or menu item is the associated part name (as shown in the part's properties dialog). The string for a menu label is the label including the shortcut key character ('&' for Win32 or '~' for OS/2) if any.

You use these strings as context strings that identify your help topics when you build your help file. For example, if you want to associate an entry field called 'NameEntryField' with a topic in an OS/2 help file, you add the following topic to your help file where 'id=' specifies the string:

...:
   :h1 id='NameEntryField'. Name EntryField Help
   Help for the name entry field.

To enable the Help Menu in your application:

1. Add a menu label part to the menu bar of your window.
2. Set its label to '&Help' (for Win32) or '~Help' (for OS/2).
3. Link the menu label's clicked event to the help menu part's popupHelpMenu message.
4. Create a link from the window's `helpManager` event to the help menu's `setHelpManager:` message.

5. If desired, link the help menu's `productInformation` event to your About dialog's `open` message.

**To add items to the Help Menu:**

If you want to have additional menu items in your help menu, you can make a copy of the help menu part by saving it under a new name, and edit the part in the Workbench and add the menu items to the menu.

Edit the external interface of the help menu part to surface an event and trigger that event by linking the `clicked` event of your new menu item to the new external event.

**Events**

A help menu part has the following event.

`productInformation`

Triggered when the user selects 'Product information' from the Help menu.

Link this event to the `open` message of your application's About dialog.

Triggered By: the user

**Messages**

A help menu part has the following messages.

`open`

A no-op; ignore this message.

Returns: —

Triggers: —

`popupHelpMenu`

Displays the help menu. Create a link from the 'Help' menu label part in the menu bar of your window to this message.

Returns: —

Triggers: —

`setHelpManager:` `aHelpManager`

Associates the help menu part with the help manager of a particular window.

Create a link from the window's `helpManager` event to this message.

Returns: —

Triggers: —
Direct-Edit

Direct-editing the help menu opens a workbench on this nested part, so you can modify it. Note that your modifications will affect the Help Menu part in the catalog, unless you save the changes in a .PAR file with a different name (in which case you will need to change the nested part in your application to use the new file).

Property Edit

See the Nested Part chapter later in this document.

Related Parts

The help menu is constructed from the following parts. Refer to their chapters (later in this document) for information about the events and messages to use when modifying the help menu.

- **Menu**
  The menu part responds to the `popup` message and serves as a pane to contain menu text items and menu separators.

- **Menu Item Separator**
  A menu separator draws a horizontal line between two menu items to separate groups of menu items.

- **Menu Text Item**
  A menu text item triggers the `clicked` event when it is chosen. Connect this event for each item you want to implement.
Example

This illustration shows a modified version of the HELPSAMP.PAR sample application, with the parts and links repositioned for greater clarity. This help-enabled application (along with a help file and its source) is located in the SAMPLE\HELP subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.

This application also demonstrates the technique of creating cascaded menus by linking the clicked event of a menu item on the primary menu to the popUp message of the secondary menu.
**Information Dialog**

An information dialog is a predefined visual part that presents a two-part message (the title of the dialog and its main text) and an OK button.

The following illustration shows an example of a Windows information dialog:

The following illustration shows an example of an OS/2 information dialog:

---

**Operation**

To operate an information dialog, click on the OK button or press the Enter key.

For the OS/2 version of this part, when the end user's system warning beep feature is enabled, the speaker beeps each time a confirmation dialog opens. To view the setting of this feature:

1. Open the OS/2 System folder.
2. Open the System Setup folder.
3. From the Sound icon's pop-up menu, choose Open/Settings.
4. Turn to the Warning Beep page of the settings notebook.

The warning beep is enabled when the Warning Beep checkbox is checked, and disabled when the checkbox is unchecked.

---

**Events**

- **closed**
  Occurs when the user clicks on the OK button, presses the Enter key, or closes the dialog via the control menu.

  Triggered By: the user
Information Dialog

Messages

open
Opens the dialog with the title and message text defined in the Properties dialog.

Returns: —
Triggers: —

openText: textString
Opens the dialog with the title defined in the Properties dialog and the message text contained in textString.

Returns: —
Triggers: —

openTitle: titleString text: textString
Opens the dialog with the title contained in titleString and the message contained in textString.

Returns: —
Triggers: —

setText: aString
Sets the dialog's text string to aString.

Returns: —
Triggers: —

setTitle: aString
Sets the title string of the dialog to aString.

Returns: —
Triggers: —

setWaitForAnswer: aBoolean
Sets the dialog's Wait for answer property to aBoolean.

Returns: aBoolean
Triggers: —

text
Returns the dialog's text string.

Returns: a String
Triggers: —

title
Returns the title string of the dialog.

Returns: a String
Triggers: —

waitForAnswer
Returns true if the dialog's Wait for answer property is set, false otherwise.

Returns: a Boolean
Triggers: —
Direct-Edit

Direct-editing the information dialog changes the part name that appears beneath the icon in the workbench.

Property Edit

Each of the part-specific controls in the information dialog's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Title

Enter the title that appears in the window label of the dialog.

Text

Enter the message that appears in the dialog.

Icon type

Select the type of icon (information, warning, error, or none) that you want from the list. The following set of illustrations shows the appearance of the various kinds of icons in Windows:
The following set of illustrations shows the appearance of the various kinds of icons in OS/2:

**Wait for answer**

The information dialog is an operating-system defined dialog and its Wait for answer property is permanently set. See the *Dialog Window* chapter earlier in this document for information on the Wait for answer property.

**Example**

This example lets you change the title and message used as arguments for the `openTitle: text:` message of an information dialog.

This example is in the file INFOXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Integer Entry Field

Integer entry fields restrict input to integer values. Integer entry fields are operated just like entry fields, except that the characters that can be entered are restricted to the following:

- digits
- plus or minus sign

If you try to enter a another type of character, the system speaker beeps. The value of an integer entry field returned by the `value` message and set with the `setValue:` message is an Integer. The value of an empty integer entry field is always nil.

Events

**aboutToChangeTo: aValue**

Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. This event is triggered only after `aValue` has been validated by the entry field's built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the `abortChange` and `retryChange` messages to respond to the proposed new value. If this event is triggered by the `clear` or `setValue:` messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its `changed:` event with `aValue` as its event value.

Triggered By:  
- the user  
- clear  
- setValue:

**changed: anInteger**

Occurs when the field loses the input focus and the contents have changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. The new contents are provided in `anInteger`.

Triggered By:  
- the user  
- clear  
- setValue:

**entered: anInteger**

Occurs when the Enter key is pressed. The contents are provided in `anInteger`.

Triggered By:  
- the user
rightClicked
Occurs when the user clicks mouse button 2 in the field.
Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the setFocus message of another part to set the tab order.
Triggered By: the user

valueException: aValue
Triggered when the argument of the setValue: message is not a valid value for the field.
Triggered By: setValue:

Messages

abortChange
Rejects the new value entered by the user and resets the entry field’s contents to the original value. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.
Returns: —
Triggers: —

add: aNumber
Returns the result of adding aNumber to the value in the field.
Returns: a Number
Triggers: —

clear
Sets the contents to nil.
Returns: —
Triggers: changed:

disable
Ignores user input. The characters in a disabled integer entry field are dimmed.
Returns: —
Triggers: —

divide: aNumber
Returns the result of dividing the value in the field by aNumber.
Returns: a Number
Triggers: —

enable
Enables the field to respond to user input.
Returns: —
Triggers: —
Integer Entry Field

**multiply: aNumber**
Returns the result of multiplying *aNumber* by the value in the field.

*Returns:* a Number  
*Triggers:* —

**retryChange**
Rejects the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the **aboutToChange:** event. Its behavior is undefined in other situations.

*Returns:* —  
*Triggers:* —

**selectAll**
Selects all the characters in the field.

*Returns:* —  
*Triggers:* —

**setFocus**
Moves the input focus to the field. Input focus is indicated by the blinking text cursor.

*Returns:* —  
*Triggers:* —

**setValue: anInteger**
Sets the contents of the field to *aValue*. If *aValue* is not an Integer or a String, it is converted to a String before it is validated. If *aValue* is not a valid Integer, a warning dialog is displayed and the contents of the field are not changed.

*Returns:* —  
*Triggers:* changed:  
*valueException:* —

**subtract: aNumber**
Returns the result of subtracting *aNumber* from the value of the integer entry field.

*Returns:* a Number  
*Triggers:* —

**value**
Returns the contents of the field (an Integer or nil).

*Returns:* an Integer  
*Triggers:* —

**valueFormatted**
Returns the contents of the field as a String.

*Returns:* a String  
*Triggers:* —
Direct-Edit

Direct-editing an integer entry field changes its value. An Invalid input dialog is displayed if you enter characters that are not allowed in the field.

Property Edit

The part-specific properties of the integer entry field are described in the following paragraphs. The OS/2 version of this part has the additional property of Alignment.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Contents

You can change the contents of the field by either direct-editing or by typing a value in this field.

Maximum number of digits

Specifies the maximum number of digits that a user can enter in the field.

Styles

- **Border**
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**
  When checked, the digits in the field can be scrolled left and right if they are not entirely visible within the field. If this is not checked, the characters cannot be scrolled. You can continue to enter characters whether or not you can see them. The default is to allow scrolling.

- **Read only**
  When checked, the contents cannot be changed by typing. The default is to allow changes.
- **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

- **Alignment**
  For the OS/2 version of this part, the alignment of the characters within the field can be set to left-aligned, centered, or right-aligned. The default is left-aligned.

### Related Parts

- **Alphabetic Entry Field**  
  Restricts input to letters.

- **Currency Entry Field**
  Restricts input to currency related characters and displays input formatted as currency.

- **Date Entry Field**
  Restricts input to digits and displays input formatted as a date.

- **Entry Field**
  A general entry field part that allows information to be entered in any format.

- **Float Entry Field**
  Restricts input to floating point numbers.

- **Phone Number Entry Field**
  Restricts input to digits and displays input in phone number format.

- **Picture Entry Field**
  Restricts input to a pattern specified in the field's properties dialog.

### Example

FORMAT.PAR, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory, demonstrates all of the formatted entry fields. It is described in the *Alphabetic Entry Field* chapter earlier in this document.
Integer Entry Field
Launch Pad

The launch pad is a nonvisual part that lets you execute an operating-system command.

You can execute a command with the launch pad any of these ways:

- Setting the command in the Properties dialog
- Issuing the **setCommand:** message followed by the **launch** message
- Passing the command string (with optional parameters) as an argument to the **launch:** message

It also allows you launch PARTS executable (EXE) files with the **launchPartFile:** message. This message allows you to split a large application into many files. Each file is brought into memory only when it is used and is automatically garbage-collected out of memory when all of its windows are closed (note, however, that its associated DLLs are not automatically unbound). Another benefit of this message is that you can use it to launch a part file multiple times to create multiple instances of the same part.

Operation

If you provide a file name with no extension, the following rules are used (in priority order) to locate the file:

1. If a file with the specified name and no extension exists on the PARTS Workbench search path, load the application from the file. The file is assumed to contain an application in its editable form.

   **NOTE:** We recommend that you always use the PAR extension when saving your application files in the editable PAR format.

2. If there is a PAR file with the specified name in a directory on the PARTS Workbench search path, load the editable application from the file. If there is a workbench window open on this file, its contents will be used as the version of the application to be loaded. This allows you to test a modification to the part before deciding whether to save it permanently, just as you can do with a nested part.

3. If there is an EXE file with the specified name in the current directory, load the non-editable application.

Using a filename without an extension minimizes the amount of work required when moving from a development environment to a runtime environment. While you are developing, the PAR file is loaded. When you create executable files, you don't have to change the file references in the launch pad; your application will automatically look for the executable file in the current working directory.
Launch Pad

Events

**launchException: aString**
Occurs when the **launch** message is executed, if the default command string has not been set in the Properties dialog or with the **setCommand**: message.

*Triggered By:* launch

Messages

**launch**
Executes the command in the default command string.

*Returns:* —
*Triggers:* launchException:

**launch: aString**
Executes the command string specified in the argument. The argument string should include the path name of the executable file and any parameters.

*Returns:* —
*Triggers:* —

**launchPartFile: aString**
Launches a PARTS Workbench file which has either a PAR or EXE extension.

*Returns:* —
*Triggers:* —

**setCommand: aString**
Sets the default command string to the argument.

*Returns:* —
*Triggers:* —

Direct-Edit

Direct-editing the launch pad part changes the part name that appears beneath the icon in the workbench.

Property Edit

The part-specific control in the launch pad’s Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Command

Enter the default command string, if any.

Example

This example allows you to execute any command entered. For instance, to open the OS/2 system text editor, type in:

**E.EXE**

This example is in the file LNCHXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A link junction part is a nonvisual part that controls the firing of links.

The link junction can:

- Cause a single message to “branch out” and fire several links
- Cause any of several messages to fire the same output link when the same action needs to be taken for several different events
- Test an argument value and trigger one of the two output events based on its value

The link junction part has two events, `triggered:` and `triggerFailed:`. Every message triggers exactly one of these two events. The `trigger` and `trigger:` messages unconditionally fire the `triggered:` event. The other messages test the argument and fire the `triggered:` event if the argument meets the necessary conditions. If the argument doesn’t meet those conditions, the `triggerFailed:` event is fired. In both cases, the argument is used for the outgoing event value.

**Operation**

To cause a single message to fire several links, connect the input link to the `trigger` or `trigger:` messages, and connect as many output links to the `triggered:` event as desired. You can force the links connected to the `triggered:` event to fire in a specific order by sequencing them. See chapter 6, *Programming with Links*, in the *Visual Smalltalk Enterprise Workbench User’s Guide* for information on sequencing links.
Link Junction

The following illustration shows the firing of several output links from a single input link:

To cause any of several input links to fire all the output links, just connect more than one input link to the link junction.

The following illustration shows the firing of a single output link from several input links:

The same configurations can be made conditional based on the values carried on the input links, using the other messages of the link junction.
Events

triggered: aValue
Occurs when the trigger or trigger: message is executed, or when any of the other messages are executed and the argument meets the required criteria. The argument to the triggering message is provided by aValue.

Triggered By:  trigger
             trigger:
             triggerIfEmpty:
             triggerIfFalse:
             triggerIfNil:
             triggerIfNotEmpty:
             triggerIfNotNil:
             triggerIfTrue:

triggerFailed: aValue
Occurs when any of the conditional messages are executed and the argument fails to meet the required criteria. The argument to the triggering message is provided by aValue.

Triggered By:  triggerIfEmpty:
                triggerIfFalse:
                triggerIfNil:
                triggerIfNotEmpty:
                triggerIfNotNil:
                triggerIfTrue:

Messages

NOTE: 1) The message argument becomes the event value of all events triggered by any of the link junction messages having an argument. 2) These messages do not have specified return values.

trigger
Causes the triggered: event to occur. The event value is nil. This message is used whenever you want to fire all the output links.

Returns:  —
Triggers:  triggered:

trigger: aValue
Causes the triggered: event to occur. The argument of the trigger: message becomes the event value of the triggered: event.

Returns:  —
Triggers:  triggered:

triggerIfEmpty: aString or aCollection
Causes the triggered: event to occur if the argument contains no items, otherwise triggerFailed: occurs.

Returns:  —
Triggers:  triggered:
          triggerFailed:
triggerIfFalse: aBoolean
Causes the triggered: event to occur if the argument is false, otherwise
triggerFailed: occurs.

Returns: —
Triggers: triggered:
triggerFailed:

triggerIfNil: aValue
Causes the triggered: event to occur if the argument is nil, otherwise
triggerFailed: occurs.

Returns: —
Triggers: triggered:
triggerFailed:

triggerIfNotEmpty: aString or aCollection
Causes the triggered: event to occur if the argument contains any items,
otherwise triggerFailed: occurs.

Returns: —
Triggers: triggered:
triggerFailed:

triggerIfNotNil: aValue
Causes the triggered: event to occur if the argument is not nil, otherwise
triggerFailed: occurs.

Returns: —
Triggers: triggered:
triggerFailed:

triggerIfTrue: aBoolean
Causes the triggered: event to occur if the argument is true, otherwise
triggerFailed: occurs.

Returns: —
Triggers: triggered:
triggerFailed:

Direct-Edit

Direct-editing the link junction part changes the part name that appears beneath the
icon in the workbench.

Property Edit

There are no special properties for the link junction part. For an explanation of any
of the standard properties, refer to Appendix C, Standard Part Properties.
Remarks

When a conditional message is used, it is helpful to change the link junction part name to indicate the purpose of the test, as shown in the following example. This makes it easier to understand conditional events when link labels are not showing.

Example

In this example, you are asked to enter your name. When you press the Enter key, the `entered:` event of the entry field triggers, passing the contents of the entry field to the `triggerIfNotEmpty:` link junction message.

If the string is empty, `triggerFailed:` occurs, opening a dialog window to remind you to enter your name. If the string is not empty, `triggered:` occurs, opening another dialog window which displays the text you entered. The OK buttons in the dialog boxes close them.

This example is in the file `LINKJUNC.PAR` in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A list pane is a visual part that provides a general selection capability.

You select an item from a list of items in a list pane by either single or double-clicking. The new selection triggers the `changed:` and `changedIndex:` events. Double-clicking on an item also triggers the `doubleClicked:` event. You can set contents of the list pane to an Array of Strings with `setList:` or you can obtain the list of items with `list`. The currently selected item is accessed with `value`, and the selection is changed with `setValueIndex:` or with `setValue`. The current selection can also be accessed or changed by the index of its position (starting from one) in the collection of strings, using `valueIndex` and `setValueIndex:`.

In Windows, a list pane normally has a scroll bar along the right-hand edge that can be used to scroll the items. If the Show disabled scroll bar property is not checked, and the entire contents of the list can be seen without scrolling, then a scroll bar is not displayed. If the Show disabled scroll bar property is checked, a disabled scroll bar will be displayed when the entire contents of the list can be seen without scrolling.

The following illustration shows the PARTS Workbench Link Color Settings dialog, which contains four list panes:

![PARTS Link Color Settings](image)

**Operation**

To select an item in the list pane, click on the item. You can use the scroll bar or arrow keys to scroll through the items, and press the Enter key to select an item when it is highlighted.
List Pane

Events

**aboutToChange**
Occurs when an item is selected. If the `abortChange` message is executed as a consequence of this event, the selection is canceled and the `changed:` event is not triggered.

*Triggered By:* the user

**changed:** aString
Occurs when the selection changes. The event value is the newly selected item.

*Triggered By:* the user

**changedIndex:** anInteger
Occurs when the selection changes. The event value is the index of the newly selected item.

*Triggered By:* the user

**doubleClicked:** aString
Occurs when the user double-clicks on an item. This action triggers the `changed:` and `changedIndex:` events first, then the `doubleClicked:` event. The event value is the newly selected item.

*Triggered By:* the user

**rightClicked**
Occurs when the user clicks on the list pane with mouse button 2.

*Triggered By:* the user

**tabbed**
Occurs when the Tab key is pressed and the list pane has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

*Triggered By:* the user

Messages

**abortChange**
Cancels the selection if executed as a consequence of triggering the `aboutToChange` message.

*Returns:* —

*Triggers:* —

**deleteAll**
Removes all items from the selection list.

*Returns:* —

*Triggers:* —
**deleteItem: aString**
Removes the item that matches `aString` from the selection list.

*Returns:* —
*Triggers:* —

**disable**
Ignores user input. The text in a disabled list pane is dimmed.

*Returns:* —
*Triggers:* —

**enable**
Enables the list pane to respond to user selections.

*Returns:* —
*Triggers:* —

**insertItem: aString**
Appends `aString` to the end of the selection list.

*Returns:* —
*Triggers:* —

**list**
Returns the items in the selection list.

*Returns:* an Array of Strings
*Triggers:* —

**setFocus**
Moves the input focus to the list pane.

*Returns:* —
*Triggers:* —

**setList: aString or anArrayofStrings**
Sets the items in the selection list to the items in the argument and returns the new list.

*Returns:* aString or anArrayofStrings
*Triggers:* —

**setValue: aString**
Sets the current selection to the first item in the selection list that contains the same text as the argument. If no item in the selection list matches the argument the selection is not changed and no events are triggered.

*Returns:* aString
*Triggers:* aboutToChange
changed:
changedIndex:
**List Pane**

**setValueIndex: anInteger**
Sets the current selection to the item whose ordinal position (starting at one) in the selection list is the argument. If the value of the argument is not between one and the number of items in the list, the selection is not changed and no events are triggered.

*Returns:* anInteger

*Triggers:* aboutToChange changed:

**value**
Returns the currently selected item; *nil* if the list is empty or nothing is selected.

*Returns:* a String

*Triggers:* —

**valueIndex**
Returns the index (starting with one) of the currently selected item; *nil* if the list is empty or nothing is selected.

*Returns:* an Integer

*Triggers:* —

**Direct-Edit**
Direct-edit lets you enter the list of items into the list pane. Use the Enter key to separate items. Click outside the list pane when you are done entering items.

**Property Edit**
Each of the part-specific controls in the list pane's Properties dialog is described in the following paragraphs. The Windows version of this part has the additional property of Show disabled scroll bar.

For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties.*
Styles

- **Horizontal scroll bar**
  Controls whether a scroll bar appears at the bottom of the list pane. The default is to not display the scroll bar.

- **Show disabled scroll bar**
  In Windows, if this box is checked, the list pane has a disabled scroll bar when it does not contain enough text to require scrolling. If this box is unchecked, the list box does not display a scroll bar if there is not enough text to require scrolling.

- **No adjust height**
  Lets the last visible item be partially displayed if it doesn’t fit entirely within the pane. If this box is checked, only those items that can be completely displayed will be. The default is to display partial items.

**Edit contents...**
Opens a text pane so you can enter or edit the item list.

**Drag drop...**
Opens the following dialog where you can enable the capability for objects to be dragged from and/or dropped on the part:

![Drag Drop Properties dialog](image)

The Source and the Target check boxes in the Drag group pane are used to enable (or disable) the part as the source and/or the target for a specified set of drag and drop operations.

The Unused and the Chosen and Prioritized list panes in the Drag Target Formats group pane are used to indicate how the part should render a dropped object, and they are disabled unless the part has been enabled as a target. This target part attempts to render the dropped object in one of the chosen formats, beginning with the first entry in the Chosen and Prioritized list pane. If the object cannot be rendered in that format, then the other chosen formats are tried. If the object cannot be rendered in any of the chosen formats, then it is ignored. To move formats...
between the Unused and the Chosen and Prioritized list panes, drag them from one list to the other. To change the priority order of the chosen formats, drag them within the Chosen and Prioritized list pane.

The Select Operations multiple-selection list pane in the Drag Target Operations group pane is used to indicate the set of drag and drop operations that the part will accept, and it is disabled unless the part has been enabled as a target. An operation is selected if it is highlighted.

The Defaults button is used to restore the part’s drag drop property to its default values.

Related Parts

- **Drop-Down List**
  A drop down list works just like a list pane except that only the current selection is visible.

- **Menu**
  A menu is a list of items that appears on demand. One item can be selected.

- **Multiple Choice List**
  A multiple choice list is a list pane that can have more than one item selected at the same time.

Example

When you click on the item in the list pane (located in the upper left corner of the window), the `changed:` and `changedIndex:` events are triggered. The event values are displayed in static text parts to the right of the event labels. If you enter an item
in the setValue: entry field, that value is selected if it is in the list. Similarly, changing the setValueIndex: entry field changes the selection to index. You can append items to the list by entering them into the insertItem: entry field.

When you select an item in the list pane, the aboutToChange event is triggered. This event is linked to LinkJunction1’s triggerIfFalse: message. If the Allow changes? check box is not checked, the link junction sends the abortChange message to the list pane and the selection does not change. You can, however, still change the selection in the list pane by using the setValue: and setValueIndex: entry fields. If the Allow changes? check box is checked, then the link junction’s triggered: event is not triggered and the change is allowed.

This example is in the file LSPNXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The list view is a visual part that can display a collection of items several different ways: in Icon view, Small Icon view, List view, or Report view.

Icon view represents each item by a 32 x 32 icon or bitmap above a text label, arranged left to right (then down), as shown in the following illustration:

![Icon view example](image)

Small Icon view represents each item by a 16 x 16 icon or bitmap to the left of a text label, arranged left to right (then down), as shown in the following illustration:

![Small Icon view example](image)
List view is similar to the Small Icon view, but with the items arranged in column order (top to bottom, then to the right), as shown in the following illustration:

![List View Example](image)

Report view (also known as Details view) is somewhat similar to the List view, but with additional information displayed in resizable columns to the right of the text label, as shown in the following illustration:

![Report View Example](image)

Each column in the Report view has a header control, which can be clicked on to perform an action such as sorting.

**Operation**

At runtime, the user can dynamically change between the various views.

In the Icon view or the Small Icon view, you can drag the items around to rearrange them.

In the Report view, you can resize the columns by dragging the dividers in the header control above the list. The column headings act as push buttons; for more information, see the *Header Control (Win32 only)* chapter earlier in this document.
Events

A list view has the following events.

**aboutToChangeLabel: anObject to: aString**
This event is triggered when the user has directly edited the contents of an item, and that change is about to be committed; this is the opportunity for validation to take place. The first event argument specifies the item that was directly edited, and the second argument specifies the new text that was entered. An event handler for this message may send the message `#abortChange` to the control to cancel the direct editing of that item.

*Triggered By:* the user

**aboutToEditLabel: anObject**
This event is triggered with the list item as an argument when that item was selected for direct editing. An event handler for this message may send the message `#abortChange` to the control to disallow direct editing of that item.

*Triggered By:* the user

**changed: anObject**
The selection was changed to that specified by the event argument.

*Triggered By:* the user

**changedLabel: anObject to: aString**
This event is triggered after a direct edit by the user was successfully completed. The first event argument is the object which was direct edited, and the second argument specifies the new text.

*Triggered By:* the user

**clicked: anObject**
This event is triggered when an item is clicked with mouse button 1; the event argument indicates the list element that was clicked.

*Triggered By:* the user

**clickedIndex: anInteger**
This event is triggered when an item is clicked with mouse button 1; the event argument indicates the index of the list element that was clicked.

*Triggered By:* the user

**columnClicked: anInteger**
This event is triggered when a column heading (in Report view only) is clicked with mouse button 1; the event argument indicates the index of the column that was clicked.

*Triggered By:* the user

**doubleClicked: anObject**
This event is triggered when an item is double-clicked with mouse button 1; the event argument indicates the list element that was double-clicked.

*Triggered By:* the user
**List View (Win32 only)**

**doubleClickedIndex: anInteger**
This event is triggered when an item is double-clicked with mouse button 1; the event argument indicates the index of the list element that was double-clicked.

*Triggered By: the user*

**needsColumnsFor: anObject**
The control needs the column data for the specified list item. The event handler should answer an IndexedCollection (such as an Array) of Strings which are the column data for the item. Note that this result must match (in both size and order) the columns that were defined by the `#headings:` message sent to the control.

**needsContents**
Occurs when the list view is about to be opened, to request that its contents be set. An event handler would respond with the `#contents:` message.

**needsImageFor: anObject**
The control needs the optional image for `anObject`. The event handler should return a 32 x 32 bitmap or icon.

**needsSmallImageFor: anObject**
The control needs the optional image for `anObject`. The event handler should return a 16 x 16 bitmap or small icon.

**needsStateFor: anObject**
The control needs the state for `anObject`. The event handler should return a value of true (for checked), false (for unchecked), or nil (for indeterminate) to indicate the item's initial state. If not handled, the item's state is assumed to be unchecked.

**needsStringFor: anObject**
The control needs the displayable string for `anObject`. The event handler should return a String. If not handled, the `#asString` message is used.

**selected: anObject**
This event is triggered when an item is selected with mouse button 1, and the selection has changed; the event argument indicates the list element that was clicked.

*Triggered By: the user*

**tabbed**
Occurs when the Tab key is pressed and the control has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

*Triggered By: the user*
Messages

A list view has the following messages.

**contents**
Answer the contents of the receiver.

*Returns:* a Collection of ListViewItems  
*Triggers:* —

**contents: aCollection**
Set the contents of the receiver to *aCollection*.

*Returns:* —  
*Triggers:* —

**disable**
Disable user selection from the list.

*Returns:* —  
*Triggers:* —

**enable**
Enable user selection from the list.

*Returns:* —  
*Triggers:* —

**headings**
Answer the receiver's collection of headings.

*Returns:* a Collection of HeaderItems  
*Triggers:* —

**headings: aCollection**
Set the headings to *aCollection* of HeaderItems.

*Returns:* —  
*Triggers:* —

**setFocus**
Moves the input focus to the list.

*Returns:* —  
*Triggers:* —

**sortAscending: aBoolean**
Set whether sorting of items in the Details view is to be done in ascending (true) or descending (false) order. Selection of which column this refers to is via **sortByColumn:** <columnIndexInteger>; the actual sorting of items is initiated via **sortItems**.

*Returns:* —  
*Triggers:* —

**sortByColumn**
Answers the index of the column that is used to sort the list items.

*Returns:* an Integer  
*Triggers:* —
List View (Win32 only)

**sortByColumn:** `anInteger`
Set the index of the column which is used to sort by to `anInteger`.

*Returns:*

*Triggers:*

**sortItems**
Sort the items using the current sort column and direction.

*Returns:*

*Triggers:*

**Direct-Edit**

Direct-editing the list view lets you enter the list of items. Use the Enter key to separate items. Click outside the list view when you are done entering items.

**Property Edit**

Each of the part-specific controls in the list view’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

**Direct Edit**

When this property is checked, the user is allowed to directly edit the text label of an item. By default, direct editing is not allowed.
Auto Arrange

When this property is checked, the items are automatically arranged by the operating system. This property is unchecked by default.

Always Show Selection

When this property is checked, the selected items in the list are shown highlighted even when the window containing the list view is not active. When this property is unchecked (the default), the selected items are shown highlighted only when the containing window is active.

Extended Selection

Support for extended (multiple) selection to the ListView common control is available. By default, when you create a ListView control, it supports single selection, and the following messages controlling selection are used:

- clearSelection
- selectedIndex
- selectedItem
- selectIndex:
- selectItem:

To enable extended (multiple) selection, send aListView extendedSelect or aListView extendedSelect: true; then the following messages can be used to query and manipulate the selection:

- clearSelections
- deselectIndex:
- deselectItem:
- selectAll
- selectedIndices
- selectItems:
- selectIndices:
- selectedItems

Note also that when in extended selection mode, the semantics of the argument passed with the clickedIndex:, clicked:, doubleClickedIndex:, and doubleClicked: events is slightly different: the argument is the index / item which was last clicked on (selected or deselected).

Use Images

When this property is checked, the icon or bitmap images supplied for the list are displayed. When this property is unchecked, only the text labels are displayed. By default, the supplied images are used.
**List**

When this radio button is selected, the items are displayed in List view. List view is similar to the Small Icon view (in representing each item by a 16 x 16 icon or bitmap to the left of a text label), but with the items arranged in column order. This is the default view.

**Icon**

When this radio button is selected, the items are displayed in Icon view. Icon view represents each item by a 32 x 32 icon or bitmap above a text label.

**Small Icon**

When this radio button is selected, the items are displayed in Small Icon view. Small Icon view represents each item by a 16 x 16 icon or bitmap to the left of a text label.

**Report**

When this radio button is selected, the items are displayed in Report view. Report view is similar to the List view (with a list of items represented by a 16 x 16 icon or bitmap to the left of a text label in the first column), but with additional text information displayed in columns to the right of the text label.
Top

When this radio button is selected, the items in Icon view or Small Icon view are aligned at the top of the pane. This is the default selection.

Left

When this radio button is selected, the items in Icon view or Small Icon view are aligned at the left of the pane.

Related Parts

- **Tree View (Win32 only)**
  The tree view is a visual part that displays a hierarchical list of items, with sublevels that can be expanded or collapsed.

Example

![Example screenshot of List View part](image)

This example demonstrates the List View part's capabilities. It illustrates how to build a user interface similar to that of the Windows 95 Explorer, using the tree view and list view controls. The user can dynamically change the view displayed by the list view, and can direct-edit both the tree view and the list view.
This example is in the file TVLVX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)

For further details about the tree view part used in this example, see the example in the Tree View (Win32 only) chapter later in this document.
A menu is a rectangular window that contains a list of choices.

A menu can either function as a pop-up menu, which appears at the cursor location, or a pull-down menu, which appears beneath a menu label in the menu bar of a window. A choice is made from the menu by clicking on one of the items in it. When an item is chosen, that item triggers the `clicked` event for that menu item, and the menu disappears.

**Operation**

The items in the menu that can be selected have either textual labels (in the case of the menu text item) or graphical labels (in the case of the menu bitmap item). Either type of item responds with the `clicked` event when chosen.

Another part, the menu item separator, appears as a solid line separating groups of menu items.

**Pop-up menus**

A pop-up menu is not visible until it receives the `popup` message, at which time it appears at the cursor location. A pop-up menu appears when the menu’s `popup` message is triggered by a link from a part other than a menu label.

The following illustration shows a pop-up menu:
Menu

Pull-down menus

The label for a pull-down menu appears in the menu bar of the top pane. Selecting the label in the menu bar causes the menu to appear beneath the label.

The following illustration shows a pull-down menu:

A pull-down menu is created by dropping the menu label part on the menu bar, then linking the menu label **clicked** event to the menu **popUp** event. Note that each menu label part must have a unique label.

Either type of menu disappears as soon as an item has been chosen. The menu also disappears without a choice being made when you click outside the menu or press the Esc key.

Using Menus in Nested Parts

When you have a Menu in a nested part and invoke it as a pull-down menu from a menu label part in the containing application, the nested part application must be built with **popUp** defined as an external message of the application. In the nested application, the external **popUp** message must be wired directly to the **popUp** message of the Menu. Similarly, the **clicked** event from the menu label part in the containing application must directly send **popUp** to the nested part containing the actual Menu part. The pull-down menu will not work properly if the **popUp** message is sent to the nested Menu indirectly (for example, if the external message in the nested application is wired to some other part, with a result link then sending **popUp** to the Menu).

Events

The menu part has no events.

Messages

**popUp**

Pops up the menu over the cursor (pop-up) or below the menu label (pull-down). You can select one of the items by clicking on it.

*Returns:* —

*Triggers:* —
Direct-Edit

Direct-editing a menu part opens a prompter that allows you to change the part name.

Property Edit

The menu part does not have any special properties. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

```
Menu Properties

Part name: Menu
Interface...

OK Cancel
```

Related Parts

- **Menu Bitmap Item**
  
  A graphical menu item that you can drop into a menu.

- **Menu Item Separator**
  
  A line that separates two adjacent menu items when you drop it between them in a menu.

- **Menu Label**
  
  A part that you put in the menu bar to activate a drop-down menu. The menu label’s `clicked` event is connected to the menu’s `popUp` event.

- **Menu Text Item**
  
  A textual menu item that you can drop into a menu.

Remarks

When you drag a menu item or menu separator over a menu, you see a line indicating the position that the item or separator will appear in if you drop it. You can adjust the position by dragging the item up or down. The menu automatically resizes itself to accommodate all of the items it contains. You can shrink the menu to conserve screen space when you are not working with it.

See the *Menu Text Item* chapter later in this document for information on creating hierarchical or cascading menus.
Example

This example contains one menu with three menu text items (labeled one, two, and three) and one menu separator. The menu can be activated by clicking on the pull-down label in the menu bar or by pressing the popup button. The other buttons allow you to see what disabled and checked items look like.

Choosing one of the menu items displays the item label in the entry field.

This example is in the file MENU.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Menu Bitmap Item

A menu bitmap item, a visual part, is a graphical menu item that displays a bitmap. When a menu bitmap item is selected from a menu, it triggers the **clicked** event.

A menu bitmap item can be disabled so that it cannot be clicked. As usual, the **disable** and **enable** messages disable and enable menu items. Menu items can be tested for whether they are disabled or not with the **isDisabled** message. If the item is disabled, the **disabledIsTrue** event is triggered by the **isDisabled** message.

The menu bitmap item can also appear with a check mark beside it to indicate that an option is in effect. The **check** and **uncheck** messages add and remove the check mark. You can test whether the item is checked with the **isChecked** message. If the item is checked, the **checkedIsTrue** event is triggered by the **isChecked** message.

The bitmap displayed can be changed at runtime with the **setBitmap:** message.

**Events**

A menu bitmap item has the following events.

- **checkedIsTrue**
  - Occurs when the **isChecked** message is executed and the menu item is checked.
  - **Triggered By:** **isChecked**

- **clicked**
  - Occurs when the user clicks on the menu item.
  - **Triggered By:** **the user**

- **disabledIsTrue**
  - Occurs when the **isDisabled** message is executed and the menu item is disabled.
  - **Triggered By:** **isDisabled**

- **hiddenIsTrue**
  - Occurs when the **isHidden** message is executed and the menu item is hidden.
  - **Triggered By:** **isHidden**
Messages

A menu bitmap item has the following messages.

**bitmap**
Returns the bitmap contained in the menu item.

*Returns:* a Bitmap

**check**
Displays a check mark beside the menu item.

*Returns:* —

**disable**
Disables the menu item so it cannot be selected.

*Returns:* —

**enable**
Enables the menu item so it can be selected.

*Returns:* —

**hide**
Hides the menu item so it is not displayed, and retains its relative position on the menu.

*Returns:* —

**isChecked**
Returns true and triggers **checkedIsTrue** if the menu item currently has a check mark displayed. Returns false otherwise.

*Returns:* a Boolean

**isDisabled**
Returns true and triggers **disabledIsTrue** if the menu item is currently disabled. Returns false otherwise.

*Returns:* a Boolean

**isHidden**
Returns true and triggers **hiddenIsTrue** if the menu item is currently hidden. Returns false otherwise.

*Returns:* a Boolean

**show**
Shows the menu item, and retains its relative position on the menu.

*Returns:* —
uncheck

Removes the check mark beside the menu item.

Returns: —
Triggers: —

setBitmap: aBitmap

Sets the bitmap to aBitmap.

Returns: aBitmap
Triggers: —

Direct-Edit

Direct-editing lets you change the menu bitmap item using the Bit Editor. See the Edit bitmap property in Appendix C, Standard Part Properties, for instructions on using the Bit Editor.

Property Edit

The menu bitmap item part does not have any special properties. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Related Parts

- **Menu**
  You connect the menu label clicked event to the menu popUp event to make a pull-down menu.

- **Menu Item Separator**
  A line that separates two adjacent menu items.

- **Menu Label**
  A part that you put in the menu bar to activate a drop-down menu. The menu label’s clicked event is connected to the menu’s popUp event.

- **Menu Text Item**
  Like a menu bitmap item, but labeled with text.

Example

See the example in the Menu chapter earlier in this document.
Menu Bitmap Item
Menu Item Separator

The menu item separator is a visual part that you drop into a menu to separate menu items into groups. The menu item separator appears in the menu as a solid horizontal line.

The following figure shows menu separators dividing four groups of menu items:

<table>
<thead>
<tr>
<th>Undo</th>
<th>Ctrl+Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Ctrl+X</td>
</tr>
<tr>
<td>Copy</td>
<td>Ctrl+C</td>
</tr>
<tr>
<td>Paste</td>
<td>Ctrl+V</td>
</tr>
<tr>
<td>Delete</td>
<td>Del</td>
</tr>
<tr>
<td>Select all</td>
<td>Ctrl+A</td>
</tr>
<tr>
<td>Print selection</td>
<td></td>
</tr>
</tbody>
</table>

Events

A menu item separator has the following event.

**hiddenIsTrue**
Occurs when the **isHidden** message is executed and the menu item separator is hidden.

Triggered By: **isHidden**

Messages

A menu item separator has the following messages.

**hide**
Hides the menu item separator so it is not displayed, and retains its relative position on the menu.

Returns: —
Trigger: —

**isHidden**
Returns true and triggers **hiddenIsTrue** if the menu item separator is currently hidden. Returns false otherwise.

Returns: a Boolean
Trigger: **hiddenIsTrue**

**show**
Shows the menu item separator, and retains its relative position on the menu.

Returns: —
Trigger: —
Menu Item Separator

Direct-Edit

Direct-editing the menu item separator opens a prompter that lets you change the part name.

Property Edit

The menu item separator part does not have any special properties. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

<table>
<thead>
<tr>
<th>MenuItemSeparator1 Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name: MenuItemSeparator1</td>
</tr>
<tr>
<td>Interface...</td>
</tr>
<tr>
<td>OK</td>
</tr>
</tbody>
</table>

Related Parts

- **Menu**
  You connect the menu label `clicked` event to the menu `popup` event to make a pull-down menu.

- **Menu Bitmap Item**
  A graphical menu item that you can drop into a menu part.

- **Menu Label**
  A part that you put in the menu bar to activate a drop-down menu. The menu label's `clicked` event is connected to the menu's `popup` event.

- **Menu Text Item**
  A textual menu item that you can drop into a menu part.

Example

See the example in the *Menu* chapter earlier in this document.
**Menu Label**

A menu label is a visual part that you drop in the menu bar of a window.

A menu label works like a button, triggering the special-purpose `clicked` event when you click on the menu label. To make a pull-down menu, connect the `clicked` event of the menu label to the `popup` event of a menu. Note that each menu label part in a menu must have a unique label.

The following illustration shows File, Edit, and Help menu labels in the menu bar of a window (with the Edit menu pulled-down).

### Operation

Drag the menu label part from the catalog and place it on the menu bar of a window part. When dropping the menu label on the menu bar, make sure that the black corner marker at the upper left corner of the menu label icon is within the menu bar.

Note that the `clicked` event is a special-purpose event used only to identify the menu part that the menu label is linked to; it is not a normal event that can be used for other purposes.

### Events

A menu label has the following event.

- **`clicked`**
  
  Occurs when the user clicks on the label in the menu bar.

  Note that this is a special-purpose event used only to identify the menu part that the menu label is linked to; it is not a normal event that can be used for other purposes.

  **Triggered By:** the user
Messages

A menu label has the following messages.

**disable**
Disables the menu label on the menu bar so it cannot be selected. The text is dimmed for a disabled menu label.

*Returns:* |

*Triggers:* |

**enable**
Enables the menu label on the menu bar so it can be selected.

*Returns:* |

*Triggers:* |

**label**
Returns the text in the menu item.

*Returns:* a String

*Triggers:* |

**setLabel:** aString
Sets the text for the menu item to aString.

*Returns:* |

*Triggers:* |

Direct-Edit

Direct-editing lets you change the text in the label.

Property Edit

The part-specific control in the menu label's Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

<table>
<thead>
<tr>
<th>MenuLabel1 Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part name: MenuLabel1</td>
</tr>
<tr>
<td>Label: uMenu</td>
</tr>
<tr>
<td>Interface...</td>
</tr>
<tr>
<td>OK</td>
</tr>
<tr>
<td>Cancel</td>
</tr>
</tbody>
</table>

Label

You can enter the text for the menu label in this field or by direct-edit. The default label is shown in the entry field.
Related Parts

- **Menu**
  You connect the menu label **clicked** event to the menu **popup** event to make a pull-down menu.

- **Menu Bitmap Item**
  A graphical menu item that you can drop into a menu.

- **Menu Item Separator**
  A line that separates two adjacent menu items when you drop it between them in a menu.

- **Menu Text Item**
  A textual menu item that you can drop into a menu.

Remarks

You can define a shortcut key that will click a specific menu label when that key is pressed. To define a shortcut key in Windows, the character just before the shortcut character in the label must be the ampersand (&) character. In OS/2, the tilde (~) character is used instead of the ampersand. The character following the ampersand or tilde becomes the shortcut key.

When the label is displayed, the ampersand or tilde is not shown, and the shortcut character is underlined to indicate that holding the Alt key and pressing the shortcut key will pop up that menu.

Example

See the example in the *Menu* chapter earlier in this document.
Menu Text Item

A menu text item is a visual part that you place into a menu part.

When the text item is selected from a menu, it triggers the **clicked** event. A menu text item can be disabled so that it cannot be clicked. Disabled menu items appear with a grayed-out label. As usual, the **disable** and **enable** messages disable and enable menu text items, but, in addition, menu items can be tested for whether they are disabled or not with the **isDisabled** message. If the item is disabled, the **isDisabled** event is triggered by the **isDisabled** message.

The menu text item can also appear with a check mark beside it to indicate that an option is in effect. The **check** and **uncheck** messages add and remove the check mark. You can test whether the item is checked with the **isChecked** message. If the item is checked, the **isChecked** event is triggered by the **isChecked** message.

Operation

When you create a link from the **clicked** event of a menu text item that does not have a label, the label is set to the name of the message sent by the link.

You can create a submenu by linking the menu text item’s **clicked** event to the **popUp** message of a menu part. An arrow is displayed to the right of a menu item when it is linked to a submenu. You **cannot** use menu bitmap items in this way.

Events

A menu text item has the following events.

- **isChecked**
  - Occurs when the **isChecked** message is executed and the menu item is checked.
  - **Triggered By:** **isChecked**

- **clicked**
  - Occurs when the user clicks on the menu item.
  - **Triggered By:** **the user**

- **isDisabled**
  - Occurs when the **isDisabled** message is executed and the menu item is checked.
  - **Triggered By:** **isDisabled**

- **isHidden**
  - Occurs when the **isHidden** message is executed and the menu item is hidden.
  - **Triggered By:** **isHidden**
Menu Text Item

Messages

A menu text item has the following messages.

check
Displays a check mark beside the menu item.

Returns:

Triggers:

disable
Disables the menu item so it can not be selected. The text is dimmed for a disabled menu text item.

Returns:

Triggers:

enable
Enables the menu item so it can be selected.

Returns:

Triggers:

hide
Hides the menu item so it is not displayed, and retains its relative position on the menu.

Returns:

Triggers:

isChecked
Returns true and triggers checkedIsTrue if the menu item currently has a check mark displayed. Returns false otherwise.

Returns: a Boolean

Triggers: checkedIsTrue

isDisabled
Returns true and triggers disabledIsTrue if the menu item is currently disabled. Returns false otherwise.

Returns: a Boolean

Triggers: disabledIsTrue

isHidden
Returns true and triggers hiddenIsTrue if the menu item is currently hidden. Returns false otherwise.

Returns: a Boolean

Triggers: hiddenIsTrue

label
Returns the text in the menu item.

Returns: a String

Triggers:

setLabel: aString
Sets the text for the menu item to the argument.

Returns:

Triggers:
**Menu Text Item**

- **show**
  Shows the menu item, and retains its relative position on the menu.
  
  **Returns:** —
  
  **Triggers:** —

- **uncheck**
  Removes the check mark beside the menu item.
  
  **Returns:** —
  
  **Triggers:** —

**Direct-Edit**

Direct-editing a menu text item lets you change the label.

**Property Edit**

Each of the part-specific controls in the menu text item's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

![MenuTextItem1 Properties](image)

### Label

You can enter the label in this field or by direct-edit.

### Accelerator

You can specify keyboard equivalents that select menu text items when the proper key combination is pressed. These equivalents appear to the right of the item label. The following illustration shows the prefabricated edit menu, which has equivalents specified for most of the menu items.
Menu Text Item

Use the check boxes and the combo box to select a keyboard accelerator for the menu item. You can select a predefined keystroke from the combo box's drop-down list or you can type a letter from A to Z in the combo box's entry field.

To define a keyboard equivalent for a menu text item by direct-editing the part:
1. Press the Tab key.
   This indicates that a keyboard equivalence combination follows.
2. Enter zero or more of the key modifier names shown in the list below.
   These keys must be pressed simultaneously with the shortcut key to activate the menu item. The names must be spelled and capitalized exactly as shown. Each modifier entered must be followed by a plus character.
   - Alt
   - Ctrl
   - Shift
3. Enter one of the following key names or letters and press the Enter key:
   - BckSp or BckSpace
   - Ins
   - Del
   - any single uppercase letter from A to Z

Related Parts

- **Menu**
  You connect the menu label `clicked` event to the menu `popup` event to make a pull-down menu.

- **Menu Bitmap Item**
  A graphical menu item that you can drop into a menu part.

- **Menu Item Separator**
  A line that separates two adjacent menu items.

- **Menu Label**
  A part that you put in the menu bar to activate a drop-down menu. The menu label's `clicked` event is connected to the menu's `popup` event.

Example

See the example in the *Menu* chapter earlier in this document.
Multiple Choice List

A multiple choice list is a visual part that provides a general selection capability, allowing more than one item to be selected at a time.

Selections are made from a list of items by clicking on them. Any time an item is selected or deselected, the changed: event is triggered. The event value is an Array containing the currently selected items.

The contents of the multiple choice list can be set to an Array of Strings with the setList: message, or the multiple choice list can be queried for the array of strings it is currently holding with the list message. The currently selected strings can be accessed with value, and the selection can be set by contents with setValue:, or by item positions with setValueIndex:. Similarly, items can be deselected by contents with deselectItem:, by position with deselectIndex:, or all selections can be cleared with deselectAll.

An item can be added to or removed from the list with insertItem: and deleteItem:.

In Windows, a multiple choice list normally has a scroll bar along the right-hand edge that can be used to scroll the items. If the Show disabled scroll bar property is not checked, and the entire contents of the list can be seen without scrolling, then a scroll bar is not displayed. If the Show disabled scroll bar property is checked, a disabled scroll bar will be displayed when the entire contents of the list can be seen without scrolling.

The following Workbench Select Parts dialog contains a multiple choice list with two parts selected:

![Image of Select Parts dialog]

Operation

Clicking once on an unselected item selects it and triggers the changed: event. Clicking again deselects it, again triggering the changed: event.
Events

aboutToChange
Occurs when a change of selection is requested. If the abortChange message is executed during the firing of any link connected to this event, the selection is canceled.

Triggered By: the user

changed: anArrayOfStrings
Occurs whenever the selection changes. The event value provides a list of the currently selected items.

Triggered By: the user
deselectAll
deselectItem:
deselectIndex:
selectIndex:
selectItem:
setValue:
setValueIndices:

rightClicked
Occurs when the user clicks on the multiple choice list with mouse button 2.

Triggered By: the user
tabbed
Occurs when the Tab key is pressed and the control has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

Messages

abortChange
Cancels a selection if executed as a consequence of triggering the aboutToChange event.

Returns: —
Triggers: —
deleteAll
Removes all items from the selection list.

Returns: —
Triggers: —
deleteItem: aString
Removes aString from the selection list.

Returns: —
Triggers: changed:
deselectAll
Deselects all selected items.

Returns: —
Triggers: changed:
**deselectIndex: anInteger**
Deselects the item whose index, counting from one at the top, is \textit{anInteger}.

- Returns: —
- Triggers: \textit{changed}:

**deselectItem: aString**
Deselects the first item that matches \textit{aString}.

- Returns: —
- Triggers: \textit{changed}:

**disable**
Ignores user selections. The text is dimmed in a disabled multiple choice list.

- Returns: —
- Triggers: —

**enable**
Enables user selection.

- Returns: —
- Triggers: —

**insertItem: aString**
Appends \textit{aString} to the selection list.

- Returns: —
- Triggers: —

**list**
Returns the items in the selection list.

- Returns: \textit{an Array of Strings}
- Triggers: —

**selectIndex: anInteger**
Selects the item whose index, counting from one at the top, is \textit{anInteger}. If there is no item with that index, the selection is not changed and no events are triggered.

- Returns: —
- Triggers: \textit{changed}:

**selectItem: aString**
Selects the first item that matches \textit{aString}. If there is no matching item, the selection is not changed and no events are triggered.

- Returns: —
- Triggers: \textit{changed}:

**setFocus**
Moves the input focus to the multiple choice list.

- Returns: —
- Triggers: —
Multiple Choice List

**setList:** aString or anArrayOfStrings
Sets the items in the selection list to the items in the argument and returns
the new list.

*Returns:* aString or anArrayOfStrings
*Triggers:* —

**setValue:** anArrayOfStrings
Sets the selection to those items matching the items in the argument.

*Returns:* —
*Triggers:* changed:

**setValueIndices:** anArrayOfIntegers
Sets the selection to the items whose indexes match those given in the
argument.

*Returns:* —
*Triggers:* changed:

**value**
Returns the currently selected items.

*Returns:* an Array of Strings
*Triggers:* —

**valueIndices**
Returns the indexes of the currently selected items.

*Returns:* an Array of Integers
*Triggers:* —

**Direct-Edit**
Direct-edit lets you to type a list of text items into the multiple choice list. Use the
Enter key to separate the items. Click outside the multiple choice list when you are
done entering items.
Property Edit

Each of the part-specific controls in the multiple choice list’s Properties dialog is described in the following paragraphs. Note that the Windows version of this part has the additional property of Show disabled scroll bar.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Styles

- **Horizontal scroll bar**
  If checked, a horizontal scroll bar is displayed at the bottom of the list. The default is to not display the horizontal scroll bar.

- **Show disabled scroll bar**
  For the Windows version of this part, if this box is checked then the list pane has a disabled scroll bar when it does not contain enough text to require scrolling. If this box is unchecked, the list box does not display a scroll bar if there is not enough text to require scrolling.

- **No adjust height**
  Lets the last visible item be partially displayed if it doesn’t fit entirely within the pane. If this box is not checked, items are displayed only if they can be displayed completely. The default is to display partial items.

**Edit contents...**

Opens a text pane so you can enter or edit the list items.
Drag drop...

Opens the following dialog where you can enable the capability for objects to be dragged from and/or dropped on the part:

The Source and the Target check boxes in the Drag group pane are used to enable (or disable) the part as the source and/or the target for a specified set of drag and drop operations.

The Unused and the Chosen and Prioritized list panes in the Drag Target Formats group pane are used to indicate how the part should render a dropped object, and they are disabled unless the part has been enabled as a target. This target part attempts to render the dropped object in one of the chosen formats, beginning with the first entry in the Chosen and Prioritized list pane. If the object cannot be rendered in that format, then the other chosen formats are tried. If the object cannot be rendered in any of the chosen formats, then it is ignored. To move formats between the Unused and the Chosen and Prioritized list panes, drag them from one list to the other. To change the priority order of the chosen formats, drag them within the Chosen and Prioritized list pane.

The Select Operations multiple-selection list pane in the Drag Target Operations group pane is used to indicate the set of drag and drop operations that the part will accept, and it is disabled unless the part has been enabled as a target. An operation is selected if it is highlighted.

The Defaults button is used to restore the part’s drag drop property to its default values.

Related Parts

- **List Pane**
  A list pane is like a multiple choice list, except that only one item can be selected at a time.
Example

![Multiple Choice List Example](image)

This example demonstrates most of the operations on a multiple choice list part. When you click on items in the multiple choice list in the upper left corner of the window, the values of the *changed:* event are displayed in the list to the right labeled “Result of changed:, value, or argument to setValue:”. This same list displays the current selections when the value button is clicked, and sets the current selection from the list when the setValue: button is clicked.

The buttons below the multiple choice list execute the messages indicated by their labels.

You can enter an item into the entry field at the top of the By Contents group, and then add it to or delete it from the item list, or select it or deselect it by pressing the corresponding button. Similarly, you can select and deselect items by position using the controls in the By Index group.

This example is in the file MULTIPLE.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Multiple Choice List
Nested Part

The nested part holder modularizes your application, making it more understandable and easier to reuse. A nested part is the mechanism for reusing a part in an application. The nested part references the part to be reused.

The referenced part may be stored in the containing application as a static snapshot (embedded reuse option) or it may be loaded as necessary from a separate file (linked reuse option). You can also use a nested part to change the reuse option of a part.

Chapter 8, *Constructing New Parts*, of the *Visual Smalltalk Enterprise Workbench User's Guide* describes how to build new parts. This part reference explains how to use a new part in an application.

Operation

This section explains the following topics:
- visual vs. nonvisual nested parts
- part reuse options
- referencing a part with a nested part
- editing the contents of a nested part
- updating a nested part from a file

Visual vs. Nonvisual Nested Parts

A nested part is either a visual or nonvisual part, depending on the part that it references:
- If the referenced part contains only nonvisual parts, then the nested part is also nonvisual.
- If the referenced part contains at least one visual part, the nested part is also visual.

Visual and nonvisual nested parts are subject to the same rules regarding placement as other visual and nonvisual parts.

When you place a visual nested part on the workbench, it is automatically shrunk. You can expand the nested part to make the visual parts it contains visible by selecting it and choosing Selected/Expand from the workbench menu bar. Expanded visual nested parts are outlined by a dotted line. Note that when you are viewing an expanded visual nested part, you can see only the visual parts that it contains; any shrunk visual parts within the nested part are not visible.
You can reshrink the part by choosing Selected/Shrink from the workbench menu bar. Visual nested parts placed in a window or dialog window are expanded and cannot be shrunk.

Although you can see the individual visual parts in an expanded, nested, visual part, you cannot select them individually. If you want to change the appearance or function of a visual nested part, you must edit the part that it references.

Shrunken nested parts represent unloaded applications. When you launch an application from the workbench, the parts referenced by its shrunken nested parts are not loaded until they are needed.

**TIP:** If the visual nested part is shrunk when you save the containing application, the application will load more quickly during development. See the *Part Accessor* chapter later in this document for information on improving the loading performance of stand-alone executable applications.

**Part Reuse Options**

When you open a new workbench, it contains a single application part. This part serves as a layer of packaging around the other parts that you add to the workbench. This package has a choice of three reuse options that affect the part when you use it in another application via a nested part. You can view and change these reuse options in the application part’s Properties dialog:

- packaged (linked)—the default setting
- packaged (embedded)
- ensemble

When you import a part, a nested part referencing the part is automatically added to the application. The part’s reuse option is indicated by the setting in the nested part’s Properties dialog.

**Ensemble Reuse Option**

The ensemble reuse option is fundamentally different from the other two options because it specifies that you don’t intend to reuse the contents of the application part as a single new part. When you add an ensemble of parts to an application, the packaging is thrown away and the parts inside are merged with the other parts in the application; the effect is the same as if you had dragged each of the parts in the ensemble from the Catalog and dropped them into the application individually.

For example, if you often use a push button paired with an entry field, you could do the following:

1. Create a part containing a push button and an entry field.
2. Set the part’s reuse option to ensemble.
3. Save the part.
4. Add the part to a Catalog (See chapter 2, *The Catalog*, in the *Visual Smalltalk Enterprise Workbench User’s Guide* for information on adding a part to a Catalog).
Nested Part

Linked Reuse Option

If the part’s reuse option is linked, the nested part maintains a link to the original file. The link allows the nested part to always reference the most current version of a part. A linked nested part must be located on the PARTS Workbench search path, otherwise a warning dialog is displayed when you attempt to add it to an application. See Setting User Preferences in chapter 1 of the Visual Smalltalk Enterprise Workbench User’s Guide for information on changing the PARTS Workbench search path.

See Updating a Nested Part from a File later in this chapter for more information on how nested parts referencing linked parts are updated.

Embedded Reuse Option

If the application’s reuse option is embedded, the nested part loads a static copy of the original part. The path of the source file is maintained for your reference but the application is not automatically updated. If you want to take advantage of updates to the original part, you must explicitly update the nested part.

NOTE: Since a link to the file is maintained for linked files, make sure that you do not rename them or remove them from your PARTS Workbench search path.

Referencing a Part with a Nested Part

When you reference a part with a nested part you can either accept the part’s default reuse option (linked) or you can explicitly set the reuse option.

To reference a part with a nested part using the part’s default reuse option:

> Drag the part’s icon from a Catalog and drop it on the workbench or in a window/dialog window as appropriate.

See Adding a Part to a Catalog Page in chapter 2 of the Visual Smalltalk Enterprise Workbench User’s Guide for information on adding a new part (application) to a Catalog.

If the part’s default reuse option is linked, it must be located on the PARTS Workbench search path.

When you drop the part’s icon on the workbench, a nested part referencing the part is automatically added to the workbench.

or

1 Choose Edit/Import part... from the workbench menu bar.

An Import Part dialog is displayed.

2 Select a part file and click on the OK button.

The cursor is replaced by the part’s icon.

If the part’s default reuse option is linked, it must be located on the PARTS Workbench search path.

3 Click on the workbench or in a window/dialog window as appropriate to place the nested part.

When you drop the part’s icon on the workbench, a nested part referencing the part is automatically added to the workbench.
To specify the reuse option when wrapping an application with a nested part:

1. Drag a nested part from the PARTS Catalog and drop it on the workbench.
2. Double-click on the nested part to open its Properties dialog.
3. Select a reuse option.

   See Part Reuse Options earlier in this chapter for an explanation of the choices. Your reuse option selection overrides the part’s default reuse option.
4. Click on the Change file... button.

   A Change file for: dialog is displayed.
5. Select a part and click on the OK button.

   If you intend to use the linked reuse option, the part you select must be on the PARTS Workbench search path.
6. Click on the OK button in the nested part's Properties dialog.

   The nested part icon is replaced by part's application icon. See the Application chapter earlier in this document for information on viewing or changing a part's application icon. If you specify the ensemble reuse option, the nested part is deleted and the contents of the referenced part are merged with the contents of the new application.
7. If the nested part is visual, you can expand it so that you can see the visual parts it contains by selecting it and then choosing Selected/Expand from the workbench menu bar.

Editing the Contents of a Nested Part

Keep the following points in mind when editing nested parts:

- When you edit the part referenced by an expanded, visual, nested part (linked or embedded), the nested part is updated to reflect any changes as soon as you switch back to the containing application's workbench.
- If you are editing any embedded part referenced by a nested part, you do not need to explicitly save changes to the part.

   Embedded parts referenced by nested parts are stored in the containing application itself rather than an external file and are saved with the containing application. If you want to change the part in the original file, you must explicitly save your changes back to the original file.
- If you are editing any linked part referenced by a nested part, you must save your changes back to the original file if you want to make them permanent.

   Linked parts referenced by nested parts are loaded from the original files when they are needed; they are not stored in the containing application. If you do not save changes, they are lost when you close the part’s workbench without saving changes.
To edit the part referenced by a nested part:

1. <Alt> click on the nested part to direct-edit it.
   A workbench is opened on the referenced part. If no part has been referenced, an empty workbench is opened.

2. Edit the part.

3. If the part’s reuse option is set to “linked” and you want to make the changes permanent, choose File/Save from the workbench menu bar.

4. In the nested part’s Properties dialog, click OK.

Updating a Nested Part from a File

Linked Parts

Whenever you update a nested part that references a linked part, a snapshot of the referenced part is stored in the containing application. This is done automatically when you open an application containing nested parts that reference linked parts. If the source file for a linked part changes, you must update the snapshot of the part to keep it in sync with the file.

You can update all the nested parts in an application by replacing the containing application (File/Replace).

You can update individual nested parts in one of several ways:

- Open the nested part’s Properties dialog and then click the OK button to close the dialog.
  The time stamp of the current snapshot is compared to the time stamp of the part’s source file. If the source is newer, a confirmation dialog is displayed asking if you want to update the part. If you click on the Yes button, the part’s snapshot is updated from the file. If you click on the No button, the part’s reuse mode is changed to embedded and the snapshot is not changed.

- Direct-edit a nested part or click on the Edit contents... button in the nested part’s Properties dialog.
  The time stamp of the current snapshot is compared to the time stamp of the part’s source file. If the source is newer, a confirmation dialog is displayed asking if you want to update the part. If you click on the Yes button, a workbench is opened on a fresh snapshot of the part taken from the file. If you click on the No button, the part’s reuse mode is changed to embedded and a workbench is opened on the snapshot stored in the containing application.

Changes you make in the linked part’s workbench are reflected in the containing application as soon as you switch back to its workbench. If you close the linked part’s workbench without saving changes, your changes are discarded and the nested part is updated from the linked part’s source file.

When you launch an application or save it as an EXE or PAX, nested parts are not updated; the current snapshots are used to avoid unexpected changes in behavior.
Embedded Parts

An embedded part is never automatically updated from its source file. You can, however, manually update it by clicking on the Change file... button in the nested part’s Properties dialog and selecting the embedded part’s source file.

You can also update an embedded part by changing the reuse mode (in the nested part’s Properties dialog) from embedded to linked if the source part is still in the PARTS Workbench search path.

When you edit an embedded part (by direct-editing the nested part or by clicking on the Edit contents... button in the nested part’s Properties dialog), changes made in the embedded part’s workbench are automatically reflected in the containing application’s workbench. Even if you close the embedded part’s workbench without saving the changes, the changes are still reflected in the containing application. If you explicitly save your changes, a confirmation dialog is displayed asking if you want to overwrite the embedded part’s source file.

To manually update the contents of an embedded nested part:

1. Double-click on the nested part containing the embedded nested part.
   The nested part’s Properties dialog is displayed.
2. Click on the Change file... button.
   A Change file for: dialog is displayed.
3. Select the source file for the nested part and click on the OK button.
4. In the nested part’s Properties dialog, click on the OK button.
   or
1. Double-click on the nested part containing the embedded nested part.
   The nested part’s Properties dialog is displayed.
2. Select the Linked reuse option.
3. In the nested part’s Properties dialog, click on the OK button.

Events

The nested part has only the external events defined by the part that it references.

Messages

In addition to the external messages defined by the part that it references, a nested part has the following message:

**open**

The external interface of all parts includes an open message by default. The message triggers the part’s **open** event which is part of its internal implementation.

When you use this message in a script, you must send it to the containing application (rather than directly to the nested part itself).

| Returns: | — |
| Triggers: | open |
Direct-Edit

When you direct-edit a nested part, a separate workbench is opened on the referenced part. If you haven’t referenced a part yet, an empty workbench is opened.

Property Edit

Each of the part-specific controls in the nested part’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

File name

The path name of the source for the part that is referenced by the nested part.

Nested part use option

The user of the nested part specifies the reuse option for the referenced application.

Change file...

Opens a “Change file for:” dialog that allows you to browse the disk and select a new part file to replace the current one.

Edit contents...

Opens a separate workbench on the application or an empty workbench if no application has been referenced yet.

Content properties...

Opens the custom Properties dialog developed for an application. (For further information, see the Property Interface chapter later in this document.)
**Nested Part**

**Related Parts**

- **Application**
  You reference an application with a nested part in order to reuse it in other applications.

- **Part Accessor**
  Analogous to a runtime nested part. Provides performance improvement for stand-alone executable applications.

**Remarks**

If you remove the source file for a nested part referencing a linked part from the PARTS Workbench search path, the containing application won't be able to use it.

For the Windows version of this part, you cannot place a nested part which contains a window into another window.

**Example**

This example uses a nested part to reference a custom-built confirmer. When the button labeled “open nested part” is pressed, the message **open** is sent to the nested part. It generates the events **OK** and **Cancel**.

This example is in the file NESTEDPA.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A notebook is a visual part that organizes and displays information in units resembling paper pages in a tabbed notebook.

A notebook has one or more notebook pages. A notebook page is a pane that contains other visual parts. A notebook page can display any text or graphics that can appear in any window. Like any other visual part, the notebook itself must be contained in a window or dialog window.

Notebook pages can have two levels of tabs – major and minor. Each tab can display either text or graphics and has properties that can be set. A notebook also has a status line that can display text describing the current page. This line appears beneath the pages in the notebook.

The notebook does not manage the behavior of any parts placed into its pages. It acts only as a container for the notebook pages.

The input focus is indicated by a dashed line around the tab of the currently selected page.

Notebook pages are identified in two different ways. Each page has a number which is the sequential order, starting with one, in which it appears in the notebook. Each page can also have a textual key value specified as a property.

Whenever the current page changes, the `changedPage:` and `changedPageNumber:` events are triggered. The `changedPage:` event provides the key of the new page as the event value. The `changedPageNumber:` event provides the number of the new page as the event value.

The notebook pages can be turned by the user or by several messages. The `turnToPage:` and `turnToPageNumber:` messages select a page by key or page number. The `firstPage`, `lastPage`, `nextPage`, and `previousPage` messages access the pages sequentially.

When you add a notebook page to a notebook, it is inserted after the current page. The page numbers are changed so that all pages are numbered in sequence.

**Operation**

You can turn the pages of a notebook with the mouse by:

> Clicking on a major or minor tab.

or

> Clicking on the right or left arrows to turn to the next or previous page.
When the notebook has the input focus, you can use the following keyboard navigation techniques:

- Press the Page Down or Page Up keys to turn to the next or previous page.
- Press the Home or End keys to turn to the first or last page.
- Press the left or right arrow keys to move between tabs, then press either the space bar or the Enter key to bring the page whose tab has the input focus to the front.
- Press the Tab key to change the focus from the major tab to the minor tab of the currently displayed page.

Note that the contents of a notebook page are in a different tab scope than that of the containing notebook. As a result, the focus must be on a control of the current page for mnemonic shortcut keys to work. Besides clicking with the mouse, you can switch focus between the notebook and the current page with the following keyboard techniques:

- To switch from the notebook to the current page, hold down the Alt key and press the down arrow.
- To switch from the current page to the notebook, hold down the Alt key and press the up arrow.

Events

**aboutToChange**

Occurs when a page is selected. If the `abortChange` message is executed as a result of this event, the page selection is canceled.

Triggered By: the user
firstPage
lastPage
nextPage
previousPage
turnToPage:
turnToPageNumber:

**changedPage:** aString

Occurs when the notebook is turned to another page. The value `aString` is the key of the newly selected page.

Triggered By: the user
firstPage
lastPage
nextPage
previousPage
turnToPage:
turnToPageNumber:
changedPageNumber: anInteger

Occurs when the notebook is turned to another page. The value anInteger is the page number of the newly selected page.

Triggered By: the user
firstPage
lastPage
nextPage
previousPage
turnToPage:
turnToPageNumber:

rightClicked

Occurs when the user clicks somewhere on the notebook (other than a notebook page) with mouse button 2. Each notebook page has its own rightClicked event.

Triggered By: the user

Tabbed

Occurs when the Tab key is pressed and the notebook has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

Messages

abortChange

Cancels a selection if executed as a consequence of triggering the aboutToChange event.

Returns: —
Triggers: —

disable

Causes the notebook to ignore user selections.

Returns: —
Triggers: —

enable

Enables the notebook to respond to user input.

Returns: —
Triggers: —

firstPage

Selects the first page of the notebook.

Returns: —
Triggers: aboutToChange
changedPage:
changedPageNumber:
lastPage

Selects the last page of the notebook.

Returns: —

Triggers: aboutToChange
  changedPage: 
  changedPageNumber:

nextPage

Turns to the page following the currently selected page.

Returns: —

Triggers: aboutToChange
  changedPage: 
  changedPageNumber:

page

Returns the key of the currently selected page.

Returns: a String

Triggers: —

pageNumber

Returns the page number of the currently selected page.

Returns: an Integer

Triggers: —

previousPage

Turns to the page preceding the currently selected page.

Returns: —

Triggers: aboutToChange
  changedPage: 
  changedPageNumber:

setFocus

Moves the input focus to the notebook.

Returns: —

Triggers: —

turnToPage: aString

Selects the page whose key is aString.

Returns: —

Triggers: aboutToChange
  changedPage: 
  changedPageNumber:

turnToPageNumber: anInteger

Changes the current page to the page whose page number is specified by anInteger.

Returns: —

Triggers: aboutToChange
  changedPage: 
  changedPageNumber
Direct-Edit

Direct-editing a notebook lets you change the extents of the major and minor tabs, and the page buttons, using the Notebook Edit sub-dialog described under the Edit Contents... property.

Property Edit

Each of the part-specific controls in the notebook's Properties dialog is described in the following paragraphs. The OS/2 version of this part has the additional properties of Back pages, Binder type, Major tabs, and Tabs shape.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Styles

- **Back pages**
  For the OS/2 version of this part, you can display the page edges of the pages behind the current page from one of four different perspectives. The position is the corner at which the two visible sides of the notebook meet.

- **Binder type**
  For the OS/2 version of this part, this property controls the appearance of the binder holding the notebook pages together.

- **Major tabs**
  For the OS/2 version of this part, you can display the major tabs on either of the visible page edges. The default is to display major tabs on the edge opposite the binding. Minor tabs are displayed on the remaining visible edge.

- **Status text alignment**
  Controls the alignment of text in the status line. The default is left-aligned.

- **Tab alignment**
  Controls the alignment of text in major and minor tabs. The default is left-aligned.

- **Tabs shape**
  For the OS/2 version of this part, this property controls the shape of both major and minor tabs.

**Demand load**
When this check box is selected, the notebook will create only the first page when it is launched, creating the rest of the pages as they are selected by the user. Check this box if you need to improve initial start-up time of your application.
Edit contents...

Brings up the Notebook Edit dialog shown in the following illustration. This dialog lets you set the major and minor tab dimensions for all pages in the notebook. The values shown in the illustration are the defaults.

- **Major tabs**
  Sets the extent of major tabs in pixels.

- **Minor tabs**
  Sets the extent of minor tabs in pixels.

- **Page buttons**
  Sets the size of the buttons used to access the previous and next page.

Related Parts

- **Notebook Page**
  Notebook pages are panes used with the notebook.

Example
This example contains a notebook that has three pages. Each page contains controls to change the selection to a different page in the notebook. In addition, the list pane contains the page keys. The notebook page can be changed by selecting one of the keys in the list pane. Whenever the selection changes, the page key and page number provided by the `changedPage:` and the `changedPageNumber:` events are displayed in the entry fields on the right.

The first page of the notebook contains buttons connected to the `lastPage` and `nextPage` messages. It has a major text tab.

The second notebook page, shown in the following illustration, lets you choose a page number with the drop-down list, then use it as the argument to the `turnToPageNumber:` message when the turn-to button is pressed.

The third and final notebook page, shown in the following illustration, has buttons connected to the notebook `previousPage` and `firstPage` messages.

This example is in the file NOTEXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The notebook page is a visual part that creates a new, blank notebook page.

When you drop a notebook page on a notebook it is placed after the current page in the notebook. An empty page window is created when you create a new notebook page. You can place other visual parts in the page window. You can turn the notebook pages when you are editing an application in the workbench, as needed, to add parts and link them. To turn the page, click on the page buttons (the arrows) at the bottom of the notebook.

A notebook page can have a major or minor tab. Major tabs define the beginning of major sections in the notebook, while minor tabs define the beginning of subsections within a major section. Major and minor tab attributes for a page are set from the Properties dialog.

A notebook page can have a status text line associated with it. The status line is displayed below the notebook page. The status text is set by direct-edit or in the Properties dialog.

Events

**rightClicked**

Occurs when the user clicks mouse button 2 on the notebook page.

Triggered By: the user

**selected**

Occurs when the user clicks mouse button 1 on the notebook page or when any of the notebook messages are used to turn the notebook to a different page.

Triggered By: the user

firstPage

lastPage

nextPage

previousPage

turnToPage:

turnToPageNumber:

Messages

A notebook page has no messages.
Direct-Edit

Direct-editing a notebook page lets you enter the status line text that is displayed below the page in the notebook when that page is showing.

Property Edit

Each of the part-specific controls in the notebook page’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Page key
Enter a unique string in this field to identify the notebook page by name. This key is used by the changedPage: event and the turnToPage: messages to identify individual notebook pages. The default is to not have a page key.

Status line text
Enter the text that is displayed on the status line when this page is showing. The default is not to display any text.

Tab
You can specify either a major tab, a minor tab, or no tab for a page. The default is no tab.

Tab style
You can select the page tab type for pages that have tabs. A page tab can be either text or a bitmap. A blank tab is displayed if you specify a tab but do not set the tab style.

To set a text tab:
1. Select the Text radio button.
2. Enter the text to appear on the tab in the entry field to the right of the radio button.
To set a bitmap tab:
1. Select the Bitmap radio button.
2. Press the Edit bitmap... button. The PARTS Workbench bitmap editor appears.
3. Either read in a bitmap file or draw the bitmap to appear on the tab and save it.

Related Parts
- Notebook
  Note book pages are placed in notebooks.
  Most visual parts can be placed onto a notebook page.

Remarks
You can define a shortcut key for a textual page tab that will turn to a specific page when the notebook has the input focus and that key is pressed. To define a shortcut key in Windows, the character just before the shortcut character in the tab text must be the ampersand (&) character. In OS/2, the tilde (~) character is used instead of the ampersand. The character following the ampersand or tilde becomes the shortcut key.

When the tab is displayed, the ampersand or tilde is not shown, and the shortcut character is underlined to indicate that it is the shortcut key for that page.

Example
See the example in the Notebook chapter earlier in this document.
Number Holder

A number holder part is a nonvisual part that serves as a variable to hold a Number. The number holder is a value holder with the class set to Number so that the messages of Number are also available. These messages perform computations and conversions on numeric values.

Operation

Numbers are values of types Integer, Float, or Fraction. A number holder part can be used as a constant or an initialized variable by setting the initial value in the Properties dialog. The initial value (zero by default) appears as the number holder icon on the workbench.

The number holder supports a variety of mathematical functions which are described in the Message table. Three messages (incrementValue, decrementValue, and changeValueBy) change the value of the receiving number holder. Other operations do not effect the value of the number holder to which they are sent. The following illustration shows several number holders on the workbench.

See the Value Holder chapter later in this document for more information about events and messages common to all value holders.

Numbers with many digits are visually truncated when displayed. An ellipsis (...) is appended to the end of the number to indicate that there is more. The full number is visible in the Properties dialog and when you direct-edit the number holder.

Screen Math

If your program needs to know the position of one screen coordinate relative to another coordinate, one solution would be to compare the values using the standard comparison operators. For example, if you want the higher of two coordinates, you could compare the y-values of the two coordinates. But is the larger value higher on the screen? Or is it the lower?
Different operating systems base screen coordinates on different origins. One operating system may calculate coordinates starting at the upper left corner of the screen while another may begin on the lower left or somewhere on the right. If the screen origin is in the upper left corner, then y=3 would be above y=5. If the screen origin is in the lower left corner then the relationship changes: y=5 is above y=3.

The number holder provides a set of messages (listed in the message table) which relate to physical positions on the screen (which are system independent) rather than an arbitrary screen origin (which is system dependent). For example 3 isAbove: 5 is true on a system where the origin is in the upper left corner while 5 isAbove: 3 is true on a system where the origin is in the lower left corner.

**Events**

A number holder has only the standard value holder events.

**Messages**

The number holder is a specialization of a value holder. In addition to the standard value holder messages, it also has the number holder specific messages listed here. Unless otherwise stated, these operations do not alter the value of the number holder part to which the message is sent.

* `aNumber`
  Multiplies the value in the number holder by `aNumber` and returns the result.
  
  Returns: a Number
  Triggers: —

+ `aNumber`
  Adds `aNumber` to the value in the number holder and returns the result.
  
  Returns: a Number
  Triggers: —

- `aNumber`
  Subtracts `aNumber` from the value in the number holder and returns the result.
  
  Returns: a Number
  Triggers: —

/ `aNumber`
  Divides the value in the number holder by `aNumber` and returns the result. When the result of integer division does not reduce to an integer, the result is of type fraction.
  
  Returns: a Number
  Triggers: —

\ `aNumber`
  Divides the value in the number holder by `aNumber` and returns the integer quotient with truncation towards negative infinity.
  
  Returns: an Integer
  Triggers: —
\( aNumber \)
Divides the value in the number holder by \( aNumber \) and returns the integer remainder with truncation towards negative infinity.

Returns: an Integer  
Triggers: —

@ \( aNumber \)
Returns a point with the value in the number holder as the \( x \) coordinate and \( aNumber \) as the \( y \) coordinate.

Returns: a Point  
Triggers: —

< \( aNumber \)
Returns true if the value of the number holder is less than \( aNumber \), otherwise returns false.

Returns: a Boolean  
Triggers: —

\( \leq \) \( aNumber \)
Returns true if the value of the number holder is less than or equal to \( aNumber \), otherwise returns false.

Returns: a Boolean  
Triggers: —

> \( aNumber \)
Returns true if the value of the number holder is greater than \( aNumber \), otherwise returns false.

Returns: a Boolean  
Triggers: —

\( \geq \) \( aNumber \)
Returns true if the value of the number holder is greater than or equal to \( aNumber \), otherwise returns false.

Returns: a Boolean  
Triggers: —

abs
Returns the absolute value of the number holder.

Returns: a Number  
Triggers: —

arcCos
Returns the arc-cosine, an angle in radians, of the value in the number holder.

Returns: a Number  
Triggers: —

arcSin
Returns the arc-sine, an angle in radians, of the value in the number holder.

Returns: a Number  
Triggers: —
Number Holder

arcTan
Returns the arc-tangent, an angle in radians, of the value in the number holder.
Returns: a Number
Triggers: —

asCharacter
Returns the character whose ASCII encoding is the value of the value in the number holder.
Returns: a Character
Triggers: —

asFloat
Returns the value in the number holder as a floating point number.
Returns: a Float
Triggers: —

between: min and: max
Returns true if the value in the number holder is greater than or equal to min and less than or equal to max, otherwise returns false.
Returns: a Boolean
Triggers: —

ceiling
Returns the integer nearest the value in the number holder towards positive infinity.
Returns: an Integer
Triggers: —

changeValueBy: aNumber
Adds aNumber (a positive or negative number) to the value in the number holder. This message changes the value stored in the number holder. It returns the new value.
Returns: a Number
Triggers: changed:

cos
Returns the cosine of the value in the number holder. The value in the number holder is the angle measured in radians.
Returns: a Number
Triggers: —

decrementValue
Decrements the value by one, and returns the new value. This message changes the value stored in the number holder.
Returns: a Number
Triggers: changed:
degreesToRadians
Returns the number of radians the value in the number holder represents in degrees.

Returns: a Number
Triggers: —

denominator
Returns 1 unless the value in the number holder is a fraction, in which case it returns the denominator of the fraction.

Returns: an Integer
Triggers: —

down: aNumber
The value in the number holder represents the y-value of an x,y coordinate on the screen. Returns the y-value that is aNumber units below the value in the number holder on the screen. For more details, see Screen Math earlier in this chapter.

Returns: a Number
Triggers: —
even
Returns true if the integer part of the value in the number holder is even, otherwise returns false.

Returns: a Boolean
Triggers: —
exp
Returns the exponential of the value in the number holder.

Returns: a Float
Triggers: —
factorial
Returns the factorial of the value in the number holder.

Returns: an Integer
Triggers: —
floor
Returns the integer nearest the value in the number holder truncating towards negative infinity.

Returns: an Integer
Triggers: —
gcd: anInteger
Returns the greatest common divisor between the value in the number holder and anInteger.

Returns: an Integer
Triggers: —
Number Holder

**higherOf**: *aNumber*

The value in the number holder and *aNumber* are numbers representing the y-values of two x,y coordinates on the screen. Returns the y-value representing the highest position on the screen. For more details, see *Screen Math* earlier in this chapter.

*Returns*: a Number

*Triggers*: —

**incrementValue**

Increments the value by one, and returns the new value. This message changes the value stored in the number holder.

*Returns*: a Number

*Triggers*: —

**isAbove**: *aNumber*

The value in the number holder and *aNumber* are numbers representing the y-values of two x,y coordinates on the screen. Returns true if the y-value in the number holder is higher on the screen than *aNumber*, otherwise returns false. For more details, see *Screen Math* earlier in this chapter.

*Returns*: a Boolean

*Triggers*: —

**isAboveEqual**: *aNumber*

The value in the number holder and *aNumber* are numbers representing the y-values of two x,y coordinates on the screen. Returns true if the y-value in the number holder is as high or higher on the screen than *aNumber*, otherwise returns false. For more details, see *Screen Math* earlier in this chapter.

*Returns*: a Boolean

*Triggers*: —

**isBelow**: *aNumber*

The value in the number holder and *aNumber* are numbers representing the y-values of two x,y coordinates on the screen. Returns true if the y-value in the number holder is lower on the screen than *aNumber*, otherwise returns false. For more details, see *Screen Math* earlier in this chapter.

*Returns*: a Boolean

*Triggers*: —

**isBelowEqual**: *aNumber*

The value in the number holder and *aNumber* are numbers representing the y-values of two x,y coordinates on the screen. Returns true if the y-value in the number holder is as low or lower on the screen than *aNumber*, otherwise returns false. For more details, see *Screen Math* earlier in this chapter.

*Returns*: a Boolean

*Triggers*: —
isLeftEqualOf: aNumber
The value in the number holder and aNumber are numbers representing the x-values of two x,y coordinates on the screen. Returns true if the x-value in the number holder is as far left or further left on the screen than aNumber, otherwise returns false. For more details, see Screen Math earlier in this chapter.

Returns: a Boolean
Triggers: —

isLeftOf: aNumber
The value in the number holder and aNumber are numbers representing the x-values of two x,y coordinates on the screen. Returns true if the x-value in the number holder is further left on the screen than aNumber, otherwise returns false. For more details, see Screen Math earlier in this chapter.

Returns: a Boolean
Triggers: —

isRightEqualOf: aNumber
The value in the number holder and aNumber are numbers representing the x-values of two x,y coordinates on the screen. Returns true if the x-value in the number holder is as far right or further right on the screen than aNumber, otherwise returns false. For more details, see Screen Math earlier in this chapter.

Returns: a Boolean
Triggers: —

isRightOf: aNumber
The value in the number holder and aNumber are numbers representing the x-values of two x,y coordinates on the screen. Returns true if the x-value in the number holder is further right on the screen than aNumber, otherwise returns false. For more details, see Screen Math earlier in this chapter.

Returns: a Boolean
Triggers: —

lcm: anInteger
Returns the least common multiple between the value in the number holder and anInteger.

Returns: an Integer
Triggers: —

left: aNumber
The value in the number holder is a number representing the x-value of an x,y coordinate on the screen. Returns the x-value that is aNumber units to the left of the value in the number holder on the screen. For more details, see Screen Math earlier in this chapter.

Returns: a Number
Triggers: —
**leftMost: aNumber**
The value in the number holder and aNumber are numbers representing the x-values of two x,y coordinates on the screen. Returns either the value in the number holder or aNumber, whichever appears closest to the left side of the screen. For more details, see Screen Math earlier in this chapter.

*Returns:* a Number

**In**
Returns the natural log of the value in the number holder.

*Returns:* a Number

**log: aNumber**
Returns that float that is the log base aNumber of the value in the number holder.

*Returns:* a Float

**lowerOf: aNumber**
The value in the number holder and aNumber are numbers representing the y-values of two x,y coordinates on the screen. Returns either the value in the number holder or aNumber, whichever appears closest to the bottom of the screen. For more details, see Screen Math earlier in this chapter.

*Returns:* a Number

**max: aNumber**
Returns the value in the number holder if it is greater than aNumber, otherwise it returns aNumber.

*Returns:* a Number

**min: aNumber**
Returns the value in the number holder if it is less than aNumber, otherwise returns aNumber.

*Returns:* a Number

**negated**
Returns the negation of the value in the number holder.

*Returns:* a Number

**negative**
Returns true if the value in the number holder is less than zero, otherwise returns false.

*Returns:* a Boolean
numerator
Returns the value in the number holder unless the value in the number holder is a fraction, in which case the numerator of the fraction is returned.

Returns: an Integer
Triggers: —

double
Returns true if the integer part of the value in the number holder is odd, otherwise returns false.

Returns: a Boolean
Triggers: —

positive
Returns true if the value in the number holder is greater than or equal to zero, otherwise returns false.

Returns: a Boolean
Triggers: —

quo: aNumber
Returns the integer quotient of dividing the value in the number holder by aNumber with truncation toward zero.

Returns: an Integer
Triggers: —

radiansToDegrees
Returns the number of degrees the value in the number holder represents in radians.

Returns: a Number
Triggers: —

raisedTo: aNumber
Returns a float that is the value in the number holder raised to the power of aNumber.

Returns: a Float
Triggers: —

raisedToInteger: anInteger
Returns the value in the number holder raised to the power of anInteger.

Returns: a Number
Triggers: —

rem: aNumber
Returns the integer remainder after dividing the value in the number holder by aNumber with truncation towards zero.

Returns: an Integer
Triggers: —
right: aNumber
The value in the number holder represents the x-value of an x,y coordinate on the screen. Returns the x-value that is aNumber units to the right of the value in the number holder on the screen. For more details, see Screen Math earlier in this chapter.

Returns: a Number
Triggers: —

rightMost: aNumber
The value in the number holder and aNumber are numbers representing the x-values of two x,y coordinates on the screen. Returns either the value in the number holder or aNumber, whichever appears closest to the right side of the screen. For more details, see Screen Math earlier in this chapter.

Returns: a Number
Triggers: —

rounded
Returns the nearest integer to the value in the number holder.

Returns: an Integer
Triggers: —

roundTo: aNumber
Returns the value in the number holder rounded to the nearest multiple of aNumber.

Returns: a Number
Triggers: —

sign
Returns 1 if the value in the number holder is greater than zero, -1 if the value in the number holder is less than zero, otherwise zero.

Returns: 1, 0, or -1
Triggers: —

sin
Returns the sine of the value in the number holder. The value in the number holder is an angle measured in radians.

Returns: a Number
Triggers: —

sqrt
Returns the square root of the value in the number holder.

Returns: a Number
Triggers: —

tan
Returns the tangent of the value in the number holder. The value in the number holder is an angle measured in radians.

Returns: a Number
Triggers: —
**truncated**

Returns the integer part of the value in the number holder.

*Returns:* an Integer

*Triggers:* —

**truncateTo: aNumber**

Returns the value in the number holder truncated (towards zero) to the nearest multiple of *aNumber*.

*Returns:* a Number

*Triggers:* —

**up: aNumber**

The value in the number holder represents the y-value of an x,y coordinate on the screen. Returns the y-value that is *aNumber* units above the value in the number holder on the screen. For more details, see *Screen Math* earlier in this chapter.

*Returns:* a Number

*Triggers:* —

**Direct-Edit**

Direct-editing a number holder lets you enter a numeric expression for the new value. Numbers with many digits are visually truncated and have an ellipsis (...) appended to the end to indicate that there are more digits. The full number is visible in the Properties dialog and when you direct-edit the number holder. When you direct edit a number holder displaying an ellipsis appended to the number, a prompter displaying the existing contents is opened. You can make any desired changes in this prompter.

**Property Edit**

The part-specific control in the number holder’s Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

**Value**

You can initialize a number holder part by entering a number into this field. The default value is zero.
Number Holder

Remarks

The value is shown in the information area when the cursor is over the number holder.
The OLE control part is a visual part that allows you to use an OLE control in a PARTS application. (An OLE control is sometimes referred to as an OCX, after the file extension that is used for its DLL file.)

A control is a visual part which is used in an application window to present information and enable user interaction with the application. Many standard controls are provided by the host operating system, such as buttons, list boxes, and entry fields. An OLE control is a custom control which is implemented as an OLE (Object Linking and Embedding) object and uses certain OLE facilities to expose its events, messages, and properties.

You use an OLE control part by connecting it to a particular OLE control. Once you have chosen the control to use, you can customize the property values of the control in your application using the control's own property dialog and create links with other parts using the control's own events and messages. The OLE control part itself provides all the connection technology that is required to use an OLE control in an application window, leaving you free to concentrate on using the specific capabilities of the control in your application.

**NOTE:** OLE controls can be obtained from many software vendors. When you install an OLE control on your system, you should familiarize yourself with the documentation provided by the control's vendor so that you understand the capabilities of the control and the properties, events, and messages that it supports.

**Operation**

After dropping the OLE control part in a window, you must connect it to a specific OLE control using the following procedure:

1. Double-click on the part to open its Properties dialog.
2. Click the Change OLE control... button to open the Choose OLE Control dialog.
3. If the desired OLE control is already registered on your system, it will appear in the list and you can simply select it and then click the OK button.
   - If the desired OLE control is installed on your system but is not registered in the system’s registration database, click the Register new control... button to open a standard file browser (searching for files with the OCX file type) where you can select it.

For further details, see the Property Edit topic later in this chapter.
OLE Control (Win32 only)

Events

There are no standard events supported by the OLE control part. After an OLE control part is connected to a specific OLE control, all the events supported by the OLE control are available.

Messages

In addition to the messages supported by the OLE control that it is connected to, the OLE control part has the following messages.

**disable**
Causes the OLE control to ignore user input.

*Returns:* —

*Triggers:* —

**editProperties**
Opens the property dialog of the connected OLE control. (This message is available when an OLE control is connected.)

*Returns:* —

*Triggers:* —

**enable**
Enables the OLE control to respond to user input.

*Returns:* —

*Triggers:* —

**setFocus**
Moves the input focus to the OLE control.

*Returns:* —

*Triggers:* —

Direct-Edit

When the OLE control part is not yet connected to an OLE control, direct-editing it opens the Choose OLE Control dialog. When an OLE control is connected, direct-editing the OLE control part opens the property dialog of the OLE control.
Property Edit

Each of the part-specific controls in the OLE control’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Change OLE control...

After connecting your OLE control part to an OLE control, the Control item displays the name of the OLE control. To change the currently connected control (or initially connect to a control), press the Change OLE control... button to display the following Choose OLE Control dialog:

The list shows the names of all the OLE controls which are registered on your system. Select the desired control from the list and then click the OK button to connect your part to that control.
If the name of the control that you want to use does not appear in the list, you can press the Register new control... button to open the standard system file dialog (shown in the following illustration) which you can use to locate the appropriate OCX file.

You may encounter this situation when you have first installed an OLE control on your system. If the control vendor’s installation program does not register the control in the system registry, you will not see its name in the list even after you have installed the OCX. In this case, you must still register the control class before you can use it.

After you choose the OCX file (using the standard file dialog shown in the previous illustration) and click on the Open button, the standard registration service that all OLE controls support is executed to register the control on your system. The name of the newly-registered control is then added to the list in the Choose OLE Control dialog and automatically selected for you. Click the OK button to accept the selection and connect your part to that OLE control.

**NOTE:** If the OCX registration does not succeed, you probably have a problem with the configuration of your system. For example, the OCX may rely on another DLL which is not available or was installed in a directory which is not on your system PATH, or it may require a newer version of a standard system file.

You must identify the cause of the configuration problem and correct it in order to use the control. After you have resolved any configuration problems, you can again attempt to connect your OLE control part using the procedures described above.

**Control properties...**

When an OLE control is connected, this button opens the property dialog of that OLE control. (When no OLE control is connected, this button is disabled.)

**Border**

If checked, a border is displayed around the OLE control. The border is displayed by default.
An ordered collection holder is a nonvisual part that serves as a variable to hold an OrderedCollection. The ordered collection holder is a value holder with the class set to OrderedCollection so that the messages of OrderedCollection are also available.

See the Value Holder chapter later in this document for information about the events and messages common to all value holders.

An OrderedCollection is a data structure that stores and retrieves values by position. Values are ordered by the sequence in which objects are added and removed. Messages allow you to add, remove, and access values from both the beginning and the end of the collection, as well as by position within the collection.

Events

Ordered collection holders have the same events as value holders.

Messages

An ordered collection holder is a specialization of a value holder. In addition to the standard value holder messages, it has a number of ordered collection specific messages listed in this section.

Messages which take aCollection as an argument will accept any type of collection (such as an Array Holder, a String Holder, or a Dictionary), as well as another OrderedCollection. Keep in mind, however, that different collections store information in ways that may not convert entirely to other types of collections. For instance a Dictionary has keys while ordered collections do not. If you add the contents of a Dictionary to an ordered collection holder, only the Dictionary values would be added, not the keys.

**add: aValue**

Adds aValue to the end of the ordered collection holder.

*Returns:* aValue

*Triggers:* —

**add: newValue after: currentValue**

Inserts newValue immediately after the element currentValue in the ordered collection holder.

*Returns:* newValue

*Triggers:* —
add: aValue afterIndex: anInteger
Inserts aValue at index position anInteger + 1 in the ordered collection holder.
Returns: aValue
Triggers: —

add: newValue before: currentValue
Inserts newValue immediately before the element currentValue in the ordered collection holder.
Returns: newValue
Triggers: —

add: aValue beforeIndex: anInteger
Inserts aValue at index position anInteger - 1 in the ordered collection holder.
Returns: aValue
Triggers: —

addAll: aCollection
Adds each element in aCollection to the end of the ordered collection holder. Returns the collection which was added.
Returns: aCollection
Triggers: —

addFirst: aValue
Inserts aValue as the first element in the ordered collection holder.
Returns: aValue
Triggers: —

addLast: aValue
Inserts aValue as the last element in the ordered collection holder.
Returns: aValue
Triggers: —

asArray
Returns the contents of the ordered collection holder translated into an Array. You can place the result into an Array Holder part by using a result link to the part's setValue: message.
Returns: an Array
Triggers: —

asSet
Returns the contents of the ordered collection holder translated into a Set, eliminating any duplicate elements. You can create a part to hold a Set by setting the class of a value holder to Set.
Returns: a Set
Triggers: —
asSortedCollection
Returns the contents of the ordered collection holder translated into a member of class SortedCollection containing the elements of the ordered collection holder sorted in ascending order. You can create a part to hold a SortedCollection by setting the class of a value holder to SortedCollection.

Returns: a SortedCollection
Triggers: —

at: anInteger
The value stored at index anInteger.

Returns: a Value
Triggers: —

at: anInteger put: aValue
Stores aValue at index anInteger.

Returns: aValue
Triggers: —

concatenate: aCollection
Creates a new OrderedCollection containing the original followed by aCollection. Collections other than OrderedCollections (such as Dictionaries) are converted to OrderedCollections before concatenation.

Returns: an OrderedCollection
Triggers: —

copyFrom: beginning to: end
Creates a new OrderedCollection containing the elements of the ordered collection holder from index position beginning through index position end.

Returns: an OrderedCollection
Triggers: —

copyReplaceFrom: beginning to: end with: aCollection
Creates a new OrderedCollection with the elements at index positions beginning through index position end replaced with the elements of aCollection. aCollection can be any type of Collection, not just an OrderedCollection.

Returns: an OrderedCollection
Triggers: —

first
Returns the first element of the ordered collection holder.

Returns: a Value
Triggers: —

includes: aValue
Returns true if the ordered collection holder contains an element equal to aValue, otherwise returns false.

Returns: a Boolean
Triggers: —
Ordered Collection Holder

**isEmpty**
Returns true if the size of the ordered collection holder is zero, otherwise returns false.

*Returns*: a Boolean

**last**
Returns the last element of the ordered collection holder.

*Returns*: a Value

**occurrencesOf: aValue**
Returns the number of elements in the ordered collection holder that are equal to aValue.

*Returns*: an Integer

**remove: aValue**
Removes the first element found that is equal to aValue from the ordered collection holder.

*Returns*: aValue

**removeAll**
Removes everything from the ordered collection holder.

*Returns*: —

**removeAll: aCollection**
Removes all the elements contained in aCollection from the receiver.

*Returns*: aCollection

**removeFirst**
Removes the first element from the collection and returns it.

*Returns*: a Value

**removeLast**
Removes the last element from the collection and returns it.

*Returns*: a Value

**reversed**
Creates a new OrderedCollection containing the elements of the original in reverse order.

*Returns*: an OrderedCollection
size

Returns the number of elements in the ordered collection holder.

Returns: an Integer

Triggers: —

Direct-Edit

Direct-editing an ordered collection holder lets you change the part name that appears beneath the icon in the workbench. It is a good idea to change the label to indicate the purpose of the ordered collection holder, just as you would use a descriptive name for a variable or a constant in another programming language.

Property Edit

Each of the part-specific controls in the ordered collection holder’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

New value

This is an entry field that lets you type an expression for a value in the same format that would be used in a script.
**Ordered Collection Holder**

**Edit value...**

Opens a dialog so you can set or edit the initial contents of the ordered collection holder. This dialog treats all values as Strings. If you want to initialize the ordered collection holder with values other than Strings, use the New value entry field.

![Image of a dialog with a list pane and buttons for adding, removing, and modifying items.]

The list pane displays the current contents of the Ordered Collection.

**To add an item to the collection:**

1. If the list pane contains any items, select the item that you want the new item to appear after.
2. Enter the expression for the value to be added.
3. Press the Add button.

**To remove an item from the collection:**

1. Select the item in the list pane.
2. Press the Remove button.

**To modify an item in the collection:**

1. Select the item and it will appear in the entry field.
2. Enter the expression for the modified value.
3. Press the Modify button to record the new value for that item.

**To remove all items from the collection:**

1. Press the Remove All... button.
2. Press the OK button in the confirmation dialog.

**Value**

Displays the current value of the ordered collection.

**Related Parts**

- **Value Holder**
  
The ordered collection holder is the same as a value holder with the value holder type set to OrderedCollection except that the class cannot be changed in the ordered collection holder.
Remarks

The contents of an ordered collection holder are displayed in the workbench’s information area when you place the cursor over the ordered collection.
Ordered Collection Holder
Part Accessor

The part accessor is a nonvisual part that allows you to load and unload portions of your application at runtime.

The size of the executable file for a major real-life system can be very large. This can cause two problems:

- long start-up time because the entire file must be loaded
- large demand on memory resources causing sluggish performance

The part accessor makes it possible to break a large executable file into multiple smaller files that you can load and unload as necessary. By delaying the load time until the part is needed, the part accessor shortens the start-up time. By unloading the part when it is no longer needed, the part accessor frees up memory.

Operation

File Referencing

A part accessor contains a reference to a part file. You choose the file to reference in the part accessor's Properties dialog. After you choose a file, the full path name for the file is displayed. This is for your reference only; the part accessor does not use this path when it is time to load the part. It searches the PARTS Workbench search path for the first file matching the file name only. The search path is used during both development and runtime applications.


If the file name specified in the part accessor's Properties dialog does not have an extension, the part accessor looks for a match based on the following criteria:

1. Find an exact match (no extension).
2. If an exact match cannot be found, find the file of the same name with a PAR extension.
3. If a match cannot be found, find the file of the same name with a PAX extension.

It is recommended that you specify a file name without an extension in the part accessor's Properties dialog. This allows you to use PAR files for development and PAX files for delivery without modifying the properties of the part accessors in your application.

You can, of course, use a specific extension, either PAR or PAX for development or PAX for runtime applications. In this case, the part accessor will use the first file in the PARTS Workbench search path that exactly matches the file name and extension.
You can use PAR files and PAX files both during development and at runtime. We recommend that you use PAR files during development because you can edit them and immediately test the change. If you need to edit a PAX, you must edit the source PAR file(s) and then recreate the PAX before you can test your changes.

We recommend that you use PAX files at runtime because they reduce the file count and file size. When you save a part as a PAX file, any nested parts that the part contains are automatically included in the PAX. For example, if you ship a part containing 10 nested parts in PAX format, you only have to ship one file. If you ship the same part in PAR format, however, you have to ship 11 files: the parent part's file and one file for each of the 10 nested parts.

PAX files are smaller than PAR files because editing information required by the workbench has been stripped out. Because the files are smaller, they load faster, increasing performance.

**NOTE:** Your delivered application must include an EXE file for the user to execute. You cannot execute a PAX file directly; only via a part accessor.

**Part Loading and Unloading**

The part accessor has two file loading options:

1. **Automatic loading**—the referenced part is automatically loaded when a message is sent to it via the part accessor. The part accessor surfaces all the external events and messages of the referenced part plus some of its own.

2. **Explicit loading**—you open the referenced part explicitly by sending a `load` message to the part accessor.

When you are finished with a part referenced by a part accessor you can explicitly unload it, freeing up memory, by sending an `unload` message to the part accessor. Unloading is always explicit; never automatic. Once unloaded, a part does not generate events or respond to messages unless you reload it either automatically or explicitly.

**Usage considerations**

A part accessor is similar to a linked nested part; both allow you to reference a part for inclusion in applications, and both maintain a link to the original file. Using the part accessor instead of a nested part results in a different file structure for your delivered application.

The primary reason for using a part accessor is to improve the performance of your runtime application by reducing start-up time and memory usage. It accomplishes this by allowing you to selectively load and unload parts at runtime.

You should use part accessors sparingly. For instance, if your application has 100 windows, you probably wouldn't want to reference each window with a separate part accessor. Neither do you need to convert every nested part to a part accessor. Used in this way, the part accessor could well become a performance drag rather than an enhancement.
The best approach to using part accessors is to build your application without them at first, using nested parts instead. You may find that your application performs well without part accessors. If your application does need performance tuning, there are some guidelines that will help you decide where to use part accessors instead of nested parts.

Parts that are good candidates for use with a part accessor may have some or all of the following characteristics:

- higher level
- relatively self contained
- large
- infrequently used

Split your application into general areas of functionality so that the users will stay in a single area for a reasonably long interval of time. For example, consider an application containing a payroll component and a general accounting component. The client’s usage patterns indicate that during any one session they will do either payroll or general accounting activities almost exclusively—only rarely will both activities be performed during a single session. If this application requires performance tuning, both the payroll and general accounting components are good candidates for referencing with part accessors based on the criteria already listed.

**Substituting a Part Accessor for a Nested Part**

Once you have identified an area of your application that requires performance tuning, you need to replace the nested part currently referencing the component with a part accessor.

**To substitute a part accessor for a nested part:**

1. Drag a part accessor from the PARTS Catalog and drop it on the workbench next to the nested part that want to replace.
2. Double click on the part accessor to open its Properties dialog.
3. Click on the Change file... button.
   
   A file dialog is displayed.
4 Select the file that is currently referenced by the nested part you are replacing.

5 In the file dialog, click on the OK button.

6 In the Properties dialog, click on the OK button.

7 Shift<drag> each link connection attached to the nested part to the part accessor.

The link is redisplayed attached to the part accessor. The part accessor should have all the same messages and events as the nested part. If a pop-up menu of event or message names is displayed after you click on the part accessor, check to be sure that the part accessor and the nested part reference the same file.

8 Once the links are transferred, delete the nested part.

Events

In addition to all the external events defined by the part that it references, a part accessor has the following events:

**aboutToUnload**

This event can be linked to messages which perform final clean-up operations, such as closing files, before the part is unloaded.

*Triggered By:* unload

**loaded**

Triggered after the referenced part is loaded.

*Triggered By:* load

**referencedWhileUnloaded**

Triggered when the part accessor receives a message intended for the referenced part and the following conditions are true: 1) the part is not loaded, 2) the part accessor's Automatic Demand Load property is not set.

*Triggered By:* a message to the referenced part

**unloaded**

Triggered after the referenced part is unloaded.

*Triggered By:* unload

Messages

In addition to all the external messages defined by the part that it references, a part accessor has the following messages:

**load**

Loads the referenced part. Loading the part does not automatically send an open message, it only loads the part. Each part accessor can only load one instance of a part. This message is ignored if the part has already been loaded. If you need to create multiple instances of a part, consider using the launch pad.

*Returns:* —

*Triggers:* loaded
unload

Disconnects the referenced part, making it available for garbage collection which frees up the memory occupied by the part.

Returns: —
Triggers: aboutToUnload
         unloaded

Direct-Edit

Direct-editing the part accessor opens a workbench on the referenced part, so you can modify it.

Property Edit

Each of the part-specific controls in the part accessor's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Filename:

The name of the referenced file. This static text field is initially blank. You set the name by clicking on the Change file... button which then opens an Open Part dialog. The referenced file must be located on the PARTS Workbench search path.

Load Option

When Automatic Demand Load is checked, the referenced part is automatically loaded when a message is sent to it via the part accessor. When Use load message is unchecked, you must load the part explicitly by sending the load message to the part accessor.

Edit contents...

When you press this button, a workbench opens for the referenced part so you can modify it.
Part Accessor

Related Parts

- **Launch Pad**
  A part accessor differs from a launch pad in three important ways:
  - The external events and messages of the referenced part are available when you use a part accessor; you cannot link to the events and messages of a part launched with a launch pad.
  - A part accessor does not automatically trigger the open message when it loads a part.
  - A part accessor can only load one instance of a part; a launch pad can load multiple independent instances of the same part.

- **Nested Part**
  A part accessor differs from a nested part in that it can explicitly load and unload the referenced file at runtime.
  The OS/2 version of this part also cannot be nested in a window or dialog window, because it is a nonvisual part.

Example

The example demonstrates an application which loads a part and opens it in response to the user clicking the Open Task One button. When the user closes the window, the part is unloaded.

This example is in the file PARTACC.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Phone Number Entry Field

Phone number entry fields restrict input to digits and display their contents formatted as phone numbers. You may only enter either seven or ten digits in a phone number entry field. If you try to enter a different type of character, the system speaker beeps and the input is disallowed.

When the phone number entry field loses focus or you change the contents with the **setValue:** message, the contents are automatically formatted as a phone number. For example, if you enter the following numbers:

```
3125551212
```

they are displayed as an area code and phone number:

```
(312) 555-1212
```

The value of a phone number entry field returned by the **value** message and set with the **setValue:** message is a String consisting of the digits only. The **valueFormatted** message returns a String that represents the contents as displayed (with formatting characters such as the parentheses and hyphens).

You can set a phone number entry field to return either an empty string or the special character null when it is empty.

**Events**

**aboutToChangeTo:** *aValue*

Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the **clear** or **setValue:** messages are executed. This event is triggered only after *aValue* has been validated by the entry field's built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the **abortChange** and **retryChange** messages to respond to the proposed new value. If this event is triggered by the **clear** or **setValue:** messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its **changed:** event with *aValue* as its event value.

*Triggered By:* the user

**clear**

**setValue:**

```
(800)
```
Phone Number Entry Field

changed: aString
Occurs when the field loses the input focus and the text has changed since the input focus entered this field, or when the clear or setValue: messages are executed. The new contents are provided in aString.

Triggered By: the user
clear
setValue:

entered: aString
Occurs when the Enter key is pressed. The contents are provided in aString.

Triggered By: the user

rightClicked
Occurs when the user clicks mouse button 2 in the field.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

valueException: aValue
Triggered when the argument of the setValue: message is not a valid value for the field.

Triggered By: setValue:

Messages

abortChange
Rejects the new value entered by the user and resets the entry field’s contents to the original value. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —

clear
Sets the contents to an empty string or null depending on the setting of the “Value when empty” property in the Properties dialog.

Returns: —
Triggers: changed:

disable
Ignores user input. The characters in a disabled phone number entry field are dimmed.

Returns: —
Triggers: —

enable
Enables the field to respond to user input.

Returns: —
Triggers: —
retryChange
Rejects the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the **aboutToChange** event. Its behavior is undefined in other situations.

Returns:  
Triggers:  

selectAll
Selects all the characters in the field.

Returns:  
Triggers:  

setFocus
Moves the input focus to the field. Input focus is indicated by the blinking text cursor.

Returns:  
Triggers:  

setValue: aValue
Sets the contents of the field to *aValue*. If *aValue* is not an Integer or a String, it is converted to a String before it is validated. If *aValue* is not a valid phone number, a warning dialog is displayed and the contents of the field are not changed.

Returns:  
Triggers:  

changed:  
valueException:

value
Returns the contents of the field, an Integer.

Returns: an Integer  
Triggers:  

valueFormatted
Returns the formatted text string displayed in the field.

Returns: a String  
Triggers:  

**Direct-Edit**

Direct-editing the phone number entry field changes its value. You see an Invalid input information dialog if you enter characters that are not allowed in the field.
Property Edit

The part-specific properties of the phone number entry field are described in the following paragraphs. The OS/2 version of this part has the additional property of Alignment. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Contents

You can change the contents of the field by either direct-editing or by typing a value in this field.

Styles

- **Border**
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**
  When checked, the characters in the field can be scrolled left and right if they are not entirely visible within the entry field. If this is not checked, the characters cannot be scrolled. You can continue to enter characters whether or not you can see them. The default is to allow scrolling.

- **Read only**
  When checked, the contents can’t be changed by typing. The default is to allow changes.

- **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

- **Alignment**
  For the OS/2 version of this part, the alignment of the characters within the field can be set to left-aligned, centered, or right-aligned. The default is left-aligned.
Related Parts

- **Alphabetic Entry Field**
  Restricts input to letters.

- **Currency Entry Field**
  Restricts input to currency related characters and displays input formatted as currency.

- **Date Entry Field**
  Restricts input to digits and displays input formatted as a date.

- **Entry Field**
  A general entry field part that accepts information in any format.

- **Float Entry Field**
  Restricts input to floating point numbers.

- **Integer Entry Field**
  Restricts input to integer numbers.

- **Picture Entry Field**
  Restricts input to a pattern specified in the field's properties dialog.

Example

FORMAT.PAR, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory, demonstrates all of the formatted entry fields. It is described in the *Alphabetic Entry Field* chapter earlier in this document.
Picture Entry Field

Picture entry fields allow you to restrict the type of character (if any) that can be input at each position in the field.

The picture entry field is a customizable entry field; you define the limits on characters that can be input and formatting characters that are displayed by default. At each position in the picture entry field, you can limit input to one of the following types of characters:

- alphabetic
- numeric
- any character (no restriction)
- alphanumeric
- logical (y, Y, n, N, t, T, f, F)
- uppercase

You can also enter formatting characters such as dashes to help visually format input. The text for an empty picture entry field that is used for a social security number would include two dashes to format the number:

```
___-__-____
```

Notice that the picture entry field contains an underscore for every character that the user must input. After you enter a number, the text looks like this:

```
123-45-6789
```

You define the picture in the picture entry fields Properties dialog. When you enter text in a picture entry field, you must enter a character for each position in the field for which a character type has been defined. If you enter too few characters, the following prompter is displayed:

![Invalid input](image)

Clicking on the Yes button returns the input focus to the field so you can correct your entry. Clicking on the No button restores the contents to the previous value.

If you try to enter a different type of character than what is specified for a particular position, the system speaker beeps.
The value of a picture entry field returned by the `value` message and set with the `setValue:` message is a String consisting of the characters entered by the user. The `valueFormatted` message returns a String that includes the entire string, including formatting characters, displayed by the field.

You can set the value of an empty picture entry field to be either null or an empty string in the field's Properties dialog.

### Events

**aboutToChangeTo: aValue**

Occurs when the entry field loses the input focus and the contents have changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. This event is triggered only after `aValue` has been validated by the entry field's built-in validation procedures, so you can use this event to execute a custom validation procedure that is even more restrictive.

If this event is triggered by the user, you can use the `abortChange` and `retryChange` messages to respond to the proposed new value. If this event is triggered by the `clear` or `setValue:` messages, you must write a script to perform the validation.

After your validation (if any) is complete, the entry field triggers its `changed:` event with `aValue` as its event value.

- **Triggered By:** the user
- **clear**
- **setValue:**

**changed: aString**

Occurs when the field loses the input focus and the text has changed since the input focus entered this field, or when the `clear` or `setValue:` messages are executed. The new contents are provided in `aString`.

- **Triggered By:** the user
- **clear**
- **setValue:**

**entered: aString**

Occurs when the Enter key is pressed. The contents are provided in `aString`.

- **Triggered By:** the user

**rightClicked**

Occurs when the user clicks mouse button 2 in the field.

- **Triggered By:** the user

**tabbed**

Occurs when the Tab key is pressed and the field has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

- **Triggered By:** the user
valueException: aValue
Triggered when the argument of the setValue: message is not a valid value for the field.

Triggered By: setValue:

Messages

abortChange
Reverts the new value entered by the user and resets the entry field’s contents to the original value. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —

clear
Sets the contents to null or an empty string depending on the setting in the field’s Properties dialog.

Returns: —
Triggers: changed:

disable
Ignores user input. The characters in a disabled field are dimmed.

Returns: —
Triggers: —

enable
Enables the field to respond to user input.

Returns: —
Triggers: —

retryChange
Reverts the new value entered by the user and sets the focus back to the entry field so the user can try again. This message can only be sent when the user triggers the aboutToChange: event. Its behavior is undefined in other situations.

Returns: —
Triggers: —

selectAll
Selects all the text in the field.

Returns: —
Triggers: —

setFocus
Moves the input focus to the field. Input focus is indicated by the blinking text cursor.

Returns: —
Triggers: —
setValue: aString
Sets the contents of the field to aString. aString will accept either inutable characters only or inutable characters plus formatting characters.

For example, if you define a picture entry field to accept two numbers followed by a dash (defined as a formatting character) and then two more numbers (11-22), aString can be formatted as either “1122” or “11-22”. If aString is not a valid value, a warning dialog is displayed and the contents of the field are not changed.

Returns: —
Triggers: changed:
  valueException:

value
Returns the contents of the field, a String.

Returns: a String
Triggers: —

valueFormatted
Returns the formatted text string displayed in the field.

Returns: a String
Triggers: —

Direct-Edit
Direct-editing the picture entry field changes its value. An Invalid input dialog is displayed if you enter characters that are not allowed in the field.

Property Edit
The part-specific properties of the picture entry field are described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Contents
You can change the contents of the currency entry field by either direct-editing or by typing a value in this field.

Value when empty
When the field is empty it can return either an empty string or the special value null.

Styles
- **Border**
  When checked, a thin solid border outlines the field. The default is to display the border.

- **Auto scroll**
  When checked, the characters in the field can be scrolled left and right if they are not entirely visible within the field. If this is not checked, the characters cannot be scrolled. You can continue to enter characters whether or not you can see them. The default is to allow scrolling.

- **Read only**
  When checked, the contents can’t be changed by typing. The default is to allow changes.

- **Password**
  When checked, an asterisk character is displayed in place of each character in the contents. This feature is usually used for security purposes such as concealing a password. The default is off.

Edit picture...
This button opens the Edit Picture dialog which lets you specify the characters that the user can enter plus any formatting characters.

Enter the characters that make up the picture into the entry field at the bottom of the dialog. For example, 999-99-9999 defines the format of a U.S. Social Security number.
Use the following characters to define the format of the picture field:

<table>
<thead>
<tr>
<th>Picture Character</th>
<th>Type</th>
<th>Characters that can be entered at this position at runtime</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>alphabetic</td>
<td>a-z, A-Z</td>
<td>any letter</td>
</tr>
<tr>
<td>9</td>
<td>numeric</td>
<td>0-9</td>
<td>any digit</td>
</tr>
<tr>
<td>X</td>
<td>character</td>
<td>all</td>
<td>any character</td>
</tr>
<tr>
<td>N</td>
<td>alphanumeric</td>
<td>a-z, A-Z, 0-9</td>
<td>any letter or digit</td>
</tr>
<tr>
<td>L</td>
<td>logical</td>
<td>y, Y, n, N, t, T, f, F</td>
<td>letters for Yes/No or True/False</td>
</tr>
<tr>
<td>!</td>
<td>upper case</td>
<td>all</td>
<td>lower case letters are converted to upper case</td>
</tr>
</tbody>
</table>

| all other characters | formatting character | none | Any other picture characters appear literally in the picture field at the position they are entered in the picture. These characters are used for formatting the characters the user enters. |

Related Parts

- **Alphabetic Entry Field**
  Restrictions input to letters.

- **Currency Entry Field**
  Restrictions input to currency related characters and displays input formatted as currency.

- **Date Entry Field**
  Restrictions input to digits and displays input formatted as a date.

- **Entry Field**
  A general entry field part that accepts information in any format.

- **Float Entry Field**
  Restrictions input to floating point numbers.

- **Integer Entry Field**
  Restrictions input to integer numbers.

- **Phone Number Entry Field**
  Restrictions input to digits and displays input in phone number format.

Example

FORMAT.PAR, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory, demonstrates all of the formatted entry fields. It is described in the *Alphabetic Entry Field* chapter earlier in this document.
Printer

The printer, a nonvisual part, lets you access your system printer.

Using the printer, you can send a string or a bitmap to a system printer using **print**. You can set the default printer name when you build your application using the Properties dialog of the printer. You can set the printer name dynamically during application execution using the **setPrinterName** or the **selectAPrinter** message of the printer.

If you print a string, it should contain the complete text to be printed, including any printer control codes or formatting codes. For example, an entire PostScript file would be sent as a single string argument to a PostScript printer.

**Event**

**error: errorString**

Occurs when there is an error in the printing process. The event value is a text description of the error. The **error** event is triggered by the following error conditions:

- The data being printed is neither a String nor Bitmap object. Triggered by the **print**, **printScreen**, and **printUserRectangle** messages.
- Restarting a print job that has already been started. Triggered by the **startJob** message.
- Ending a print job that has not been started. Triggered by the **endJob** message.

**Triggered By:** beginJob, endJob, print, printScreen, printUserRectangle

**Messages**

**beginJob**

Begins a print job.

**Returns:** —

**Triggers:** error:

**defaultPrinterName**

Returns the name of the default printer.

**Returns:** a String

**Triggers:** —
endJob
Ends the current print job.

Returns: —
Triggers: error:

formFeed
Sends a form feed (page eject) to the printer.

Returns: —
Triggers: —

print: aString or aBitmap
Prints the argument on the printer. A string argument should already be
formatted with the appropriate printer control codes.

Returns: —
Triggers: error:

printerName
Returns the currently selected printer. Returns the default printer name if
none has been selected.

Returns: a String
Triggers: —

printerNames
Returns a list of all installed printer names.

Returns: a Collection
Triggers: —

printerSetup
Opens a dialog that allows you to select a printer and adjust its settings.

printScreen
Prints the entire screen on the printer.

Returns: —
Triggers: error:

printUserRectangle
Prints a user-selected rectangle on the printer.

Returns: —
Triggers: error:

selectAPrinter
Lets the user select a printer name from a list of known printers.

Returns: —
Triggers: —

selectAFont
Opens a Font selection dialog to let the user select from the available printer
fonts.

Returns: a Font
Triggers: —
setPrinterName: aString

Sets the name of the default printer to aString.

Returns: —

Triggers: —

Direct-Edit

Direct-editing the printer changes the part name that appears beneath the icon in the workbench.

Property Edit

Each of the part-specific controls in the printer’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Printer name

This is a list containing the known printer names on the system. You set the default printer name by choosing an item from the list.

Font...

Opens the standard font editor, which displays a list of the available fonts for the selected printer and allows you to choose the font used by the printer.

Remarks

If the argument to print: is not a string or a bitmap, nothing is printed.
Example

This example opens a file dialog when you press the Print file... button. Selecting a file sends the name to a file accessor's fileText: message. The result, a string containing all the characters in the file, is used as the argument for the printer's print: message.

This example is in the file PRNTXMP.LPAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Progress Bar

A progress bar is an output-only control (usually placed at the bottom of a window) that an application can use to display the progress of a lengthy operation. A progress bar is especially useful for operations that affect an entire file, such as converting, opening, saving, and printing.

The progress bar appears as a rectangle that is gradually filled from left to right as the operation progresses. While a progress bar can have any integer value for its minimum, maximum, and initial values, the default values for these properties represent the typical usage for displaying the percentage of the operation that has been completed.

Operation

The progress bar does not permit any user input.

You can decide how to update the display. One approach is to display regularly spaced milestones (such as every 10%) when they occur; another approach is to display the progress at regularly timed intervals (such as every few seconds).

To display the increasing percentage of the operation that has been completed, use the `incrementPosition` or `incrementPositionBy:` messages, and edit the values of the Range properties so that:

- Position = 0
- Minimum = 0 (the default)
- Maximum = 100 (the default)

To display the decreasing percentage of the operation that remains to be completed, use the `decrementPosition` or `decrementPositionBy:` messages, and edit the values of the Range properties so that:

- Position = 100
- Minimum = 0 (the default)
- Maximum = 100 (the default)

Events

A progress bar has the following event.

`changed: anInteger`

The current position was changed.

Triggered By: `setValue:`
Progress Bar

Messages

A progress bar has the following messages.

**decrementPosition**
Decrement the current position by the current line increment.

*Returns:*
—

*Triggers:*
—

**decrementPositionBy: anInteger**
Decrement the current position by the specified amount.

*Returns:*
—

*Triggers:*
—

**incrementPosition**
Increment the current position by the current line increment.

*Returns:*
—

*Triggers:*
—

**incrementPositionBy: anInteger**
Increment the current position by the specified amount.

*Returns:*
—

*Triggers:*
—

**lineIncrement**
Answer the current line increment.

*Returns:*
an Integer

*Triggers:*
—

**lineIncrement: anInteger**
Set the current line increment.

*Returns:*
—

*Triggers:*
—

**maximum**
Answer the maximum value of the range.

*Returns:*
an Integer

*Triggers:*
—

**maximum: anInteger**
Sets the maximum value of the range.

*Returns:*
—

*Triggers:*
—

**minimum**
Answer the minimum value of the range.

*Returns:*
an Integer

*Triggers:*
—

**minimum: anInteger**
Sets the minimum value of the range.

*Returns:*
—

*Triggers:*
—
**minimum: anInteger maximum: anInteger**
Set the minimum and maximum values of the range.

*Returns:*  
*Triggers:*  

**minimum: anInteger maximum: anInteger position: anInteger**
Set the minimum and maximum values of the range, as well as the initial position.

*Returns:*  
*Triggers:*  

**position**
Answers the current position.

*Returns:*  
*Triggers:*  

**position: anInteger**
Sets the initial position to the specified integer value, which must be within the range defined by the minimum and maximum values.

*Returns:*  
*Triggers:*  

**setValue: anInteger**
Sets the current position to the specified integer value, and trigger the `changed:` event if the specified value is different from the previous value.

*Returns:*  
*Triggers:* `changed:`

**value**
Answers the current position.

*Returns:*  
*Triggers:*  

**Direct-Edit**
There is no direct-editing for the progress bar. If you attempt to direct-edit this part, you receive the following dialog:
Property Edit

Each of the part-specific controls in the progress bar's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

![Progress Bar Properties Dialog](image)

**Position**

A non-negative integer that is the initial position. This value must be within the progress bar's range, as defined by the minimum and maximum values. The default value is 20.

**Minimum**

A non-negative integer (which must be less than the maximum value) that is the minimum value of the progress bar's range. The default value is 0.

**Maximum**

A positive integer (which must be greater than the minimum value) that is the maximum value of the progress bar's range. The default value is 100.

**Line Increment**

A positive integer that is the amount of an individual increment within the progress bar's range. The default value is 1.
Related Parts

- **Track Bar**
  The track bar is a window containing a slider control, which the user can move to select a single value within a range of integer values. A track bar has tick marks indicating the increments within its range, and can be oriented either horizontally or vertically.

Example

This example demonstrates the Progress Bar part’s capabilities.

This example is in the file RANGEX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the links repositioned for greater clarity.)

For further details about the track bar part used in this example, see the example in the **Track Bar** chapter later in this document.
The prompter, a visual part, is a predefined dialog that gets a line of text from the user.

The prompter has properties for the message and default answer, or you can pass them as arguments to the `prompt:default:` message. An example prompter is shown in the following illustration:

![Prompter Illustration](image)

**Operation**

Pressing either the OK button or the Enter key triggers the `ok:` event with the text in the entry field as the event value. Pressing the Cancel button triggers the `cancel` event.

**Events**

- **ok: aString**
  
  Occurs when the OK button or the Enter key is pressed. The value `aString` is the contents of the entry field.

  *Triggered By:* the user

- **cancel**

  Occurs when the cancel button is pressed.

  *Triggered By:* the user

**Messages**

- **defaultMessage**

  Returns the default message string.

  *Returns:* a String

- **prompt**

  Opens the prompter with the message and default answer set in the Properties dialog. Returns the contents of the entry field as a String.

  *Returns:* a String
```text
prompt: messageString default: answerString

Opens the prompter with the message messageString and default answer answerString.

Returns: —
Triggers: —

promptDefault: answerString

Opens the prompter with the default answer answerString. Returns the contents of the entry field.

Returns: a String
Triggers: —

promptMessage

Returns the prompt string.

Returns: a String
Triggers: —

removeBlanks

Returns true if the Remove Blanks property is on, false if it is not.

Returns: a Boolean
Triggers: —

setDefaultMessage: aString

Sets the default message to aString.

Returns: —
Triggers: —

setPromptMessage: aString

Sets the prompt to aString.

Returns: aString
Triggers: —

setRemoveBlanks: aBoolean

Sets the Remove Blanks property to aBoolean.

Returns: —
Triggers: —

setWaitForAnswer: aBoolean

Sets the Wait for Answer property to aBoolean.

Returns: —
Triggers: —

waitForAnswer

Returns true if the Wait for Answer property is on, false if it is not.

Returns: a Boolean
Triggers: —
```
Direct-Edit

Direct-editing the prompter changes the part name that appears beneath the icon in the workbench.

Property Edit

Each of the part-specific controls in the prompter’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Prompt
Enter the message text to be displayed when the prompt message is executed.

Default
Enter the default answer to be shown in the entry field when the prompter is brought up.

Remove leading and trailing blanks
If checked, any spaces before or after the response will be deleted before the event value is passed. These spaces are removed by default.

Wait for answer
See the Dialog Window chapter earlier in this document for a description of this property.
Example

This example opens the prompter and displays either the user's answer or the text “Canceled” depending on which prompter event is triggered. This example is in the file PROMPTER.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A property constructor is a nonvisual part used to interactively construct a property manager for an application. A property manager is a mechanism that provides a way to retrieve or modify the values of the named properties of an object. The property manager can also specify different editors or formats for the various property values.

After a property manager is constructed, the property interface part can be used to automatically generate an initial version of the Properties dialog that is the user interface to the property manager. In Windows, the generated Properties dialog is a property sheet (Win32 only) part with a visual part for each named property. In OS/2, the generated Properties dialog is a dialog window. You can then edit this generated part to improve its appearance or replace some of the generated default parts (such as replacing a set of check boxes with a group of radio buttons).

**Operation**

**To interactively build a property manager:**

1. After building your application, add a property constructor part to its workbench.
2. Open the property constructor's Properties dialog to create named properties for your application. (The standard editors and data formats are available in drop-down lists.) For each of your named properties, a `get` and a `set` event is generated when you click the OK button to close the dialog.
3. Link each `get` event (of the property constructor part) to a message (of any component in your application) that returns the value for the property, and link each `set` event to a message that sets the value for the property.
4. Open the property constructor’s Properties dialog again and, in the Property editor file field, enter the name for the automatically generated part file that will implement the application's Properties dialog.
5. Press the Edit button, which will open a new workbench and automatically generate a new property dialog. The part file will contain a property interface part and a property sheet part with a visual part for each named property.
6. Edit this file as needed to improve its appearance or replace some of the generated default parts (such as replacing a set of check boxes with a group of radio buttons).

Whenever you use the original application as a nested part in another application, this custom Properties dialog is available by pressing the Content properties... button that appears in the Properties dialog of the nested part. When the user changes a property of the application using this custom Properties dialog, the property interface part passes the changes back to the property constructor part, which then sets the values of the changed properties.
Events

A property constructor has generated `get` and `set` events for each of the named properties in your application.

Messages

A property constructor has the following message.

**constuctedPropertyHandler**

Answers the constructed property manager.

Returns: a PropertyManager

Triggers: —

Direct-Edit

Direct-editing the property constructor changes the part name.

Property Edit

Each of the part-specific controls in the property constructor's Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*. 
Property editor file

This text entry field gives the name of a “property editor” .PAR file that the property constructor will automatically generate. This file will contain a property interface part and a Properties dialog part with a visual part for each named property.

You can then edit this file to improve its appearance or replace some of the generated default parts (such as replacing a set of check boxes with a group of radio buttons).

When the user changes a property of the application using this property sheet, the property interface part passes the changes back to the property constructor part, which then updates the values of the various properties through the prelinked set events.

Edit

This button is used to edit the specified Property editor file, or to open a new workbench and automatically generate the file if it does not yet exist. Note that pressing this button closes the current Properties dialog.

Properties

This combo box allows you to select an existing property from the list by clicking on it.

To create a new property, enter a unique name in this entry field, specify either an Editor or a Format, and press the Add button.

To remove an existing property, select the property from the list and press the Remove button.

Editor

This drop-down list box is used to select a standard editor for a property. (Note that you can select either an Editor or a Format, but not both.) The following table explains the standard editors that are available.

<table>
<thead>
<tr>
<th>List Item</th>
<th>Standard Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>file name</td>
<td>The standard file dialog (for your operating system) opens, so you can select a file.</td>
</tr>
<tr>
<td>bitmap</td>
<td>The bitmap editor opens, so you can select an existing bitmap file or draw a new bitmap.</td>
</tr>
<tr>
<td>icon</td>
<td>The icon editor opens, so you can select an existing icon file or draw a new icon.</td>
</tr>
<tr>
<td>size window</td>
<td>The Window Layout dialog opens, so you can specify the location and size of the part. (To get the location and size, use the partGetWindowSize message. To set the location and size, use the partSetWindowSize: message. To access these messages, press the Other messages... button of the Create Link dialog.)</td>
</tr>
<tr>
<td>color</td>
<td>The color dialog (for your operating system) opens, so you can select a color. (To get the color, use the foreColor or backColor message. To set the color, use the foreColor: or backColor: message. To access these messages, press the Other messages... button of the Create Link dialog.)</td>
</tr>
</tbody>
</table>
Property Constructor

List Item | Standard Editor
--- | ---
font | The font dialog (for your operating system) opens, so you can select a font. (To get the location and size, use the font message. To set the location and size, use the font: message. To access these messages, press the Other messages... button of the Create Link dialog.)

Format

This drop-down list box is used to select a standard data format for a property, which determines the default visual part that is generated for the Properties dialog. (Note that you can select either an Editor or a Format, but not both.)

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Generated Visual Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>Currency Entry Field</td>
</tr>
<tr>
<td>List</td>
<td>List Pane</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer Entry Field</td>
</tr>
<tr>
<td>String</td>
<td>Entry Field</td>
</tr>
<tr>
<td>Phone</td>
<td>Phone Number Entry Field</td>
</tr>
<tr>
<td>Float</td>
<td>Float Entry Field</td>
</tr>
<tr>
<td>Boolean</td>
<td>Check Box</td>
</tr>
</tbody>
</table>

Related Parts

- **Property Interface**
  The property interface is a nonvisual part that provides the interface between an application’s property manager and its Properties dialog. The property interface part can automatically generate a Properties dialog for each named property in the application.

- **Property Sheet (Win32 only)**
  The property sheet is a visual part that simulates the Win95 property sheet control. A Win95 property sheet is a modeless dialog window containing one or more overlapping tabbed pages where you can view and edit the properties of an object.
Example

This example (located in \SAMPLE\PARTDEMO\PROPDEM.PAR) demonstrates the use of the Property Constructor and Property Interface parts, as nested parts within this application. The PROPMGR1.PAR nested part contains the property constructor part and the visual parts of the customer application.
The following illustration (of a slightly modified version of PROPMGR1.PAR, with the links repositioned for greater clarity) shows the links used to get and set the values of the various visual parts.

The PROPMGR2.PAR part contains a property interface part and a property sheet part with the visual parts used to display and set the various properties of the application. (The illustration shows a slightly modified version of this file, with the links repositioned for greater clarity.)
If you double-click on the PROPMGR1 part, the standard Properties dialog for a nested part is displayed. Click on the Content properties... button to display the following custom Properties dialog for the application.

When the user changes a property of the application using this property sheet, the property interface part passes the changes back to the property constructor part for use in the application.

This example is in the file PROPMDEM.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
A property interface is a nonvisual part that provides the interface between an application's property manager and its Properties dialog. The property interface part can automatically generate a Properties dialog for each named property in the application, in a “property editor” .PAR file. You can then edit this file to improve its appearance or replace some of the generated default parts (such as replacing a set of check boxes with a group of radio buttons).

When the user changes a property of the application using this property sheet, the property interface part passes the changes back to the property manager for use in the application.

A property manager is a mechanism that provides a way to retrieve or modify the values and formats of the named properties of an application. The property manager is either created interactively using a property constructor part or defined in Smalltalk using the `constructPropertyManager` class method. Based on this property manager, the property interface part generates `initX` events for each property to provide the initial values for the Properties dialog and `setX` messages for each property to collect the user's input. The values collected from the user are temporarily stored within the property interface part until either the Apply or the OK button is pressed, causing the values to be stored back in the application.

**Operation**

When used together, the property constructor and property interface parts enable you to interactively build a custom Properties dialog for an application that will be reused as a nested part within another application. When used alone, the property interface part enables you to build a custom Properties dialog for a primitive part.

**To interactively build a custom Properties dialog:**

1. After building your application, add a property constructor part to its workbench.
2. Open the property constructor's Properties dialog to create named properties for your application. (The standard editors and data formats are available in drop-down lists.) For each of your named properties, a `get` and a `set` event is generated when you click the OK button to close the dialog.
3. Link each `get` event (of the property constructor part) to a message (of any component in your application) that returns the value for the property, and link each `set` event to a message that sets the value for the property.
4. Open the property constructor's Properties dialog again and, in the Property editor file field, enter the name for the automatically generated part file that will implement the application's Properties dialog.
5. Press the Edit button, which will open a new workbench and automatically generate a new property dialog. The part file will contain a property interface part and a property sheet part with a visual part for each named property.
6 Edit this file as needed to improve its appearance or replace some of the generated default parts (such as replacing a set of check boxes with a group of radio buttons).

Whenever you use the original application as a nested part in another application, this custom Properties dialog is available by pressing the Content properties... button that appears in the Properties dialog of the nested part. When the user changes a property of the application using this custom Properties dialog, the property interface part passes the changes back to the property constructor part.

To build a custom Properties dialog for a primitive part:
1 Open a new workbench and drag a Property Interface part into it.
2 Open its property dialog by double-clicking the part.
3 Fill in the Object expression field with a Smalltalk expression that evaluates to the object that needs the property dialog to be built. For example, you would type ‘PARTSStatusWindowPart new’ if you are building the dialog for a Status Window part.
4 Check the Generate property dialog check box, and click on the OK button. After you click the OK button, the dialog window will be automatically generated according to the property manager returned by the constructPropertyManager method of the PARTSStatusWindowPart.
5 Edit this dialog as needed to improve its appearance or change its contents, and save it with the file name specified in the argument of the addGroupEditor: message within the constructPropertyManager method. For example, the file name that should be used for the new property dialog part for a Status Window part is ‘ppstatwi.par’.

For an additional discussion of creating a custom property edit dialog, see the “Creating a Part in Smalltalk” chapter of the Visual Smalltalk Enterprise User’s Guide.

Events

The property interface part has the following events, including an initX event (that provides the initial value for the Properties dialog) that is generated for each property defined by the property manager.

changed
Occurs whenever the user changes a value, causing a generated setX message to be executed.

Triggered By: the user

changed
Occurs whenever the user changes a value, causing a generated setX message to be executed.

Triggered By: the user

initX: aValue
Occurs whenever the underlying object is set. The argument is the initial value of the property.

Triggered By: initializeObject:
stored
Occurs after all the changed property values have been stored back in the original object.

Triggers By: storePropertyValues:

WindowTitle: aString
Occurs whenever the underlying object is set. The argument aString is the proposed title for the Properties dialog.

Triggers By: initializeObject:

Messages
The property interface part has the following messages, including a setX message (for collecting the user’s input from the Properties dialog) that is generated for each property defined by the property manager.

editPropertyFile: aFileName
Open a workbench on the specified file, which will serve as the property dialog for the receiver’s object.

Returns: —
Triggers: —

evaluate: anExpression
Evaluate the expression.

Returns: result of evaluating anExpression
Triggers: —

initializeObject: anObject
Sets the underlying object whose properties are to be edited.

Returns: —
Triggers: WindowTitle:

setX: aValue
Temporarily stores the new value of the property (in the property interface part) until the user presses either the Apply or the OK button, causing the values to be stored back in the underlying object.

Returns: —
Triggers: changed

storePropertyValues
Store the edited properties back to the object. This is usually done when the property edit is finished, but it can also be done if there is an “Apply” button.

Returns: —
Triggers: stored

supportedEditors
Answer all the editors currently supported by the receiver.

Returns: a Collection of editors
Triggers: —
supportedValidators

Answer all the validators currently supported by the receiver.

*Returns:* a Collection of validators

*Triggers:* 

**Direct-Edit**

Direct-editing the property interface changes the part name.

**Property Edit**

Each of the part-specific controls in the property interface’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

![Property Interface](image)

**Application file name**

This text entry field gives the name of the application’s .PAR file. (If the object does not currently exist, you can use the Object expression entry field.)

**Browse**

If the application file name does not appear in the list, you can use the Browse button to open the standard system file dialog which you can use to locate the appropriate file.

**Object expression**

Fill in this field with a Smalltalk expression that evaluates to the object that needs the property dialog to be built. For example, you would type ‘PARTSStatusWindowPart new’ if you are building the dialog for a Status Window part.

The display area below this entry field shows the object that is the result of evaluating the Smalltalk expression. (If the object already exists as a .PAR file, you can use the Application file name entry field.)
Generate property dialog
When checked, this property causes the generation of a Properties dialog for the selected application.

Note that this will create the dialog from scratch, eliminating any customizations that you had made previously.

Update object from file/expression
When checked, this property updates the property interface's generated $initX$ events and $setX$ messages. This updating is needed after you have changed the property manager, to reflect those changes in this property interface part.

This updating uses the value from the Application file name or the Object expression entry field, and provides you an opportunity to preserve the modifications you made to the Properties dialog after it was initially generated. Be sure that the Generate property dialog property is not checked, and manually update the existing Properties dialog to account for all the changes made in the property manager.

Related Parts

- **Property Constructor**
  The property constructor is a nonvisual part used to interactively construct a property manager for an application. A property manager is a mechanism that provides a way to retrieve or modify the values of the named properties of an object.

- **Property Sheet (Win32 only)**
  The property sheet is a visual part that simulates the Win95 property sheet control. A Win95 property sheet is a modeless dialog window containing one or more overlapping tabbed pages where you can view and edit the properties of an object.
Example

This example demonstrates the use of the Property Constructor and Property Interface parts, as nested parts within an application (located in \SAMPLE\PARTDEMO\PROPMDEM.PAR). The PROPMGR2.PAR file implements the Properties dialog for PROPMGR1.PAR, so it contains a property interface part and a property sheet part with the visual parts used to display and set the various properties of the application. The previous illustration (of a slightly modified version of PROPMGR2.PAR, with the links repositioned for greater clarity) shows the links between the property interface part and the property sheet.
The following illustration (of a slightly modified version of PROPMGR2.PAR, with the links repositioned for greater clarity) shows the links between the property interface part and the visual parts representing the named properties of the application.

This example is in the file PROPMDEM.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.

For additional explanation of this example, see the Property Constructor chapter earlier in this document.
Property Sheet  
(Win32 only)

A property sheet is a visual part that simulates the Win95 property sheet control. A Win95 property sheet is a dialog window containing a tab control which has one or more overlapping tabbed pages where you can view and edit the properties of an object. Note that the property sheet is a modeless dialog that floats on top of its parent window without disabling it.

The property sheet part consists of a tab control page (labeled “General”) in a tab control within a dialog window (labeled “Properties”) having three push buttons (labeled “OK”, “Cancel”, and “Apply”). You can add more tab control pages for other groups of properties. On each page, you place the various visual parts needed to view and edit the properties of your part or application.

The following illustration shows how a new property sheet appears in the workbench.

**Operation**

The Apply button is initially disabled, until some user input has occurred on any of the pages in the property sheet. After the enabled Apply button is pressed, it is disabled again until the next user input.

When the OK or the Apply button is pressed, the `apply` event is triggered for each page in the property sheet. When the OK or the Cancel button is pressed, the window is closed.

An optional Help button can be manually added to the page, as shown in the example in the *Tab Control (Win32 only)* chapter later in this document. When the Help button is pressed, the `help` event is triggered for the current page.
Events

See the Dialog Window, Push Button, Tab Control (Win32 only), and Tab Control Page (Win32 only) chapters in this document.

Messages

See the Dialog Window, Push Button, Tab Control (Win32 only), and Tab Control Page (Win32 only) chapters in this document.

Direct-Edit

See the Dialog Window, Push Button, Tab Control (Win32 only), and Tab Control Page (Win32 only) chapters in this document.

Property Edit

See the Dialog Window, Push Button, Tab Control (Win32 only), and Tab Control Page (Win32 only) chapters in this document.

Related Parts

- **Dialog Window**
  The dialog window is a pop-up window that can be used for a specific interaction with the user.

- **Property Constructor**
  The property constructor is a nonvisual part used to construct a property manager for an application.

- **Property Interface**
  The property interface is a nonvisual part that provides the interface between a part's property manager and its property dialog.

- **Push Button**
  The push button is a visual part with a text label describing its function, that is operated by clicking on it. Pressing a push button usually initiates an action, such as recalculating a result, canceling an operation, or accepting a new configuration.

- **Tab Control (Win32 only)**
  The tab control is a visual part, similar to a notebook, that holds tab control pages having a labeled tab that can be selected to bring it to the front for viewing.

- **Tab Control Page (Win32 only)**
  The tab control page is a visual part which is used within a tab control (a container similar to a notebook), and has a labeled tab that can be selected to bring it to the front for viewing.
Remarks

When building the property sheet for a part, besides designing the layout you are also responsible for collecting the user's input and updating the underlying part. The property constructor and property interface parts can help make these tasks easier. The property interface part can even automatically generate an initial layout of the items on the property sheet. For further information, see the Property Constructor and Property Interface chapters earlier in this document.

Example

This example demonstrates a modified version (with the links repositioned for greater clarity) of the actual property sheet for the Tab Control Page part. The various properties are defined using the Property Interface part, with related properties presented on three different pages.
The General page is used for the standard properties that are common to most parts. The following illustration shows the links used on this page.

To move between the pages of the property sheet when you are building it, perform the following steps:

1. Double-click on either the tab control part or the tab control page part of your property sheet to open a Properties dialog.
2. If the Page page (as shown in the following illustration) is not currently displayed, click on the Page tab.
3. Click on the desired page to select it, then click either the OK or the Apply button.
The following illustration shows the links used on the Tab page.

The following illustration shows the links used on the Page page.

This example is in the file PPTABPG.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Push Button

A push button is a visual part, with a label describing its function, that is operated by clicking on it or by pressing the Spacebar or the Enter key when the button has the input focus.

Pressing a push button usually initiates an action, such as recalculating a result, canceling an operation, or accepting a new configuration.

The following illustration shows two push buttons in a confirmation dialog:

![Illustration of push buttons in a confirmation dialog]

Operation

To operate a push button, click on it or, if it has input focus, press the Spacebar or the Enter key. The border of the push button changes to appear as if the button is depressed. The clicked event is triggered when you release the mouse button.

Events

clicked
Occurs when the user clicks on the button with mouse button 1, or presses the Enter key or the Spacebar when the button has the input focus.

Triggered By: the user

rightClicked
Occurs when the user clicks on the button with mouse button 2.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the push button has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user
Push Button

Messages

disable
  Causes the button to ignore user input and dims the button label.
  Returns: —
  Triggers: —

able
  Enables the button to respond to user input and restores the original
  appearance of the button label.
  Returns: —
  Triggers: —

label
  Returns the button label.
  Returns: a String
  Triggers: —

setFocus
  Moves the input focus to the push button. Input focus is indicated by a
  dashed rectangle around the label.
  Returns: —
  Triggers: —

setLabel: aString
  Sets the button label to aString.
  Returns: aString
  Triggers: —

Direct-Edit

Direct-editing a push button lets you enter or revise the button label. The button
label contains text that is displayed on the button in both the workbench and the
runtime application.

Property Edit

Each of the part-specific controls in the push button’s Properties dialog is described
in the following paragraphs. The OS/2 version of this part has the additional
property of Border.
For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Label
The initial text for the button label can be entered either in this field or by direct-editing the part.

Styles
- **Default button**
  This property applies to buttons in dialog windows only. When the focus is not on a push button, a drawn button, or text pane and you press the Enter key, the default push button or drawn button (if there is one) will be activated as if it were pressed. If the focus is on any push button or drawn button, it automatically becomes the default button even though it does not have the Default button style. You can use the arrow keys to get to such a button, and you will notice that the highlighting of the button moves from the previous default button to this new one. Now, if you press either the Space bar or the Enter key, you will select the push button or drawn button.

- **Border**
  For the OS/2 version of this part, when this property is checked, the button is displayed with a border. This is the default style.

Related Parts
- **Dialog Window**
  The dialog window is a modal pop-up window that can be used for a specific interaction with the user.

  Note that default push buttons can only be used in a dialog window.

- **Drawn Button**
  A push button whose label is a bitmap instead of text.
Push Button

Remarks

When a link is made between an event of an unlabeled button and a message, the button label is automatically set to the name of the message at the other end of the link. This default button label can then be edited or deleted.

Designating a default button only has an effect if the button is contained in a dialog window.

You can define a shortcut key that will click a specific push button when that key is pressed. To define a shortcut key in Windows, insert the ampersand (&) just before the shortcut character in the label. In OS/2, the tilde (~) character is used instead of the ampersand. The character following the ampersand or tilde becomes the shortcut key.

When the label is displayed, the ampersand or tilde is not shown, and the shortcut character is underlined to indicate that it is the shortcut key for that push button. If the part that currently has the input focus expects keyboard input (like an entry field), you need to press and hold the Alt key and then press the shortcut key.

Example

This example uses four push buttons. The bottom three buttons function as controls for the fourth button – enabling, disabling, and setting focus to it. When you press Push Me, the label is displayed in the top right entry field. You can enter a new label for the button in the upper left entry field. The Enter key must be pressed after the new label text is typed in order to trigger the entered: event.

This example is in the file PSHBXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Radio Button

A radio button is a visual part that toggles on and off to select one alternative from several.

By convention, radio buttons are mutually exclusive within a group. When there are several automatic radio buttons in a pane and they are in the same group (as specified in the containing window’s Order children... dialog), only one of the buttons can be on at a time. Turning on one radio button in a group turns all the other automatic radio buttons in the same group off.

When a radio button is pressed, it triggers the clicked: and turnedOn events. The radio button label is provided as the event value of clicked: so you can tell which radio button was selected if several radio buttons are linked to the same message.

The Boolean value of a radio button can be obtained or set using the value and setValue: messages respectively. The initial state of a radio button can be set in the Properties dialog.

The following illustration shows radio buttons being used to select text properties:

Operation

There are two styles of radio buttons: automatic and non-automatic.

Clicking on an automatic radio button turns it on and triggers the clicked: and turnedOn events. All other radio buttons in the same group are turned off.

Clicking on a non-automatic radio button triggers the clicked: event, but does not turn it on or trigger the turnedOn event. You must send either the setValue: message with a true argument or the turnOn message to turn on a non-automatic radio button. Non-automatic radio buttons allow selections to be validated before they are accepted.

The Boolean value of either kind of radio button can be changed with the setValue:, turnOn, or turnOff messages. The appearance of the radio button changes if necessary to match the new value.
Radio Button

Events

clicked: aString
Occurs when the user clicks on the radio button, regardless of the current value of the button. The event value provided is the button label. If the radio button style is automatic, it turns on and triggers the turnedOn event.

Triggered By: the user

rightClicked
Occurs when the user clicks on the radio button with button 2.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the button has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

turnedOff
Occurs when the radio button is turned off by the setValue: or turnOff messages.

Triggered By: setValue: turnOff

turnedOn
Occurs when the radio button is clicked or is turned on by the setValue: or turnOn messages.

Triggered By: the user setValue: turnOn

valueIsOn
Occurs when the turnOn, setValue:, or value message is executed and the radio button is turned on.

Triggered By: turnOn: value setValue:

Messages

disable
Deactivates the radio button so that it does not respond to user input. A disabled radio button appears with the label dimmed.

Returns: —
Triggers: —

enable
Enables the radio button so that it responds to user input and restores the original appearance.

Returns: —
Triggers: —
Radio Button

label
Returns the radio button label.

Returns: a String
Triggers: —

setFocus
Moves the input focus to the radio button. Input focus is indicated by a dashed rectangle around the label.

Returns: —
Triggers: —

setLabel: aString
Sets the label to aString.

Returns: aString
Triggers: —

setValue: aBoolean
Sets the current value of the radio button to the argument. If the value is set to true for an automatic button, all other radio buttons in the same group have their value set to false. Either turnedOff or turnedOn is triggered.

Returns: aBoolean
Triggers: turnedOff
turnedOn
valueIsOn

turnOff
Sets the radio button to off.

Returns: —
Triggers: turnedOff

turnOn
Sets the radio button to on.

Returns: —
Triggers: turnedOn
valueIsOn

value
Returns the value of the radio button – true if on, false if off. If on, the valueIsOn event is triggered.

Returns: a Boolean
Triggers: valueIsOn

Direct-Edit

Direct-editing the radio button lets you to change the label. The label is a text string that is displayed next to the button circle. If the new label is longer than the previous one, the radio button part changes size so the entire label is visible.
Radio Button

Property Edit

Each of the part-specific controls in the radio button’s Properties dialog is described in the following paragraphs. Note that the Windows version of this part has the additional property of Label left of button.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Label

You can enter the text for the radio button label in this field or with direct-edit.

Styles

- **Auto radio button**
  The default style is automatic. An automatic radio button turns on when the user clicks on it. It triggers the `clicked:` and `turnedOn` events, and turns off all the radio buttons in the same group.

  A radio button that is not automatic only changes value when the `setValue:`, `turnOn`, or `turnOff` messages are executed. Whether the circle appears filled or not depends on the value. If the value is true, the circle appears filled. Nonautomatic radio buttons respond to mouse or keyboard input with the `clicked:` event, but do not turn on or trigger the `turnedOn` event. A nonautomatic radio button can be used when the selection must be verified before it can take place.

- **Label left of button**
  For the Windows version of this part, you can choose to place the label to the left of the button. The default is to put the label to the right of the button.

Initially on

The initial value of the radio is determined by this control. Only one radio button in a group should have this control checked. The default initial value is off.
Radio Button

Related Parts

- **Check Box**
  Check boxes let you choose more than one alternative at a time.

- **Group Pane**
  A group pane provides a label and a visual grouping of radio buttons.

- **Window, Dialog Window, and Notebook Page**
  The Order children... button in the Properties dialog of these parts provides the way to specify more than one radio button group within the same window.

Remarks

Rather than disabling a group of radio buttons, consider putting them in a dialog top pane and open it when they are needed.

You can define a shortcut key that will click a specific radio button when that key is pressed. To define a shortcut key in Windows, insert the ampersand (&) character just before the shortcut-key character. In OS/2, the tilde (~) character is used instead of the ampersand. The character following the ampersand or tilde becomes the shortcut key.

When the label is displayed, the ampersand or tilde is not shown and the shortcut character is underlined to indicate that it is the shortcut key for that radio button. If the part that currently has the input focus expects keyboard input (like an entry field), you need to press and hold the Alt key and then press the shortcut key.
At the top of this example is a group pane containing two radio buttons. These radio buttons operate independently of the radio buttons outside the group pane. The push buttons send messages to the Green radio button. The entry field displays the label of the Red or Green button when one of them is clicked.

Below them is a non-automatic radio button that can be clicked, but doesn't change value unless you click the check box to its right.

This example is in the file RADIOBTN.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Report Writer

Report writer parts can be used to format, print, and display data from a PARTS Workbench application.

The parts for the PARTS Report Writer are provided in a separate catalog named RW.CAT. They are explained in the *Visual Smalltalk Enterprise Workbench Report Writer User's Guide.*
A rich edit part is a control where you can enter and edit text. The text can have rich text formatting (RTF) applied at both the paragraph and character level, and can even include embedded OLE objects. This part implements the Win95 Rich Edit control.

The rich edit part supports almost all the functions provided by the other multiline text edit controls, so applications that currently use the other controls can easily be modified to use this rich edit control.

A prebuilt menu of formatting commands is provided by the Format Menu (Win32 only) part, which is described earlier in this document.

Operation

The rich edit part provides support for printing and for streaming in and out of either rich text format or plain text. The contents can be read from and stored to a file or changed at runtime with messages or user actions. It has a modified flag that is set whenever the text changes. This flag can be tested and cleared.

Text is entered the same as in an entry field, except that pressing the Enter key just starts a new line of text. Entering or modifying text sets the modified flag. You can test to see if the contents have been modified by sending the `isModified` message. If the contents have been modified, the `modifiedIsTrue` event is triggered. After the contents have been saved, the `clearModified` message can be used to reset the modified flag.

There are two different ways to save the changes:

- Detect whether the contents have changed by sending the `isModified` message when the pane is closed.
  
  If the `modifiedIsTrue` event triggers, the contents have been modified. The `modifiedIsTrue` event can be linked to either the `value` or `saveToFile:` messages to preserve the changes.

- Save the changes is to link the `changed:` event to a message.

  The `changed:` event triggers whenever the input focus leaves the pane and changes have been made. The event value provided is a string containing the new contents. The modified flag is cleared whenever the `changed:` event triggers the execution of a message. If the `changed:` event is not linked to a message, the modified flag is not cleared.

If these two techniques for recording changes are used together, the `modifiedIsTrue` event will only be triggered if the `isModified` message is executed after changes have been made, but before the input focus leaves.
The initial contents can be formatted using the Format menu of the window displayed for direct-editing or after pressing the Edit contents... button on the General page of the Properties dialog. To enable the user of your application to modify the format of the text in the rich edit part during runtime, include a Format Menu (Win32 only) part and link its menu items to the corresponding messages of the rich edit part:

<table>
<thead>
<tr>
<th>Format Menu Item</th>
<th>Rich Edit message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font</td>
<td>changeFont</td>
</tr>
<tr>
<td>Color</td>
<td>changeColor</td>
</tr>
<tr>
<td>Paragraph</td>
<td>changeParagraph</td>
</tr>
<tr>
<td>Tabs</td>
<td>changeTabs</td>
</tr>
<tr>
<td>Increase size</td>
<td>growTextLimit</td>
</tr>
</tbody>
</table>

Note that the Increase Size command is needed because the system does not automatically increase the capacity of the rich edit control when its limit is exceeded. The initial size limit is 32K characters.

Events

A rich edit part has the following events.

**changed**: aString

Occurs when the pane loses input focus and the contents have been changed since it received the input focus, or when a message is executed that replaces the contents. The event value is the current contents. The modified flag is cleared whenever this event triggers the execution of a message.

Triggered By: the user
clear
setFromFile:
setValue:

**controlTabbed**

Occurs when the Control and Tab keys are pressed simultaneously and the pane has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

**fileException**: aString

Occurs when a file cannot be accessed. The event value contains the pathname of the file.

Triggered By: saveToFile:
setFromFile:

**modifiedIsTrue**

Occurs when the pane is sent the isModified message and the modified flag is set.

Triggered By: isModified
rightClicked
Occurs when the user clicks mouse button 2 on the pane.

Triggered By: the user

Messages
A rich edit part has the following messages.

append: aString
Appends the argument to the text in the pane and sets the modified flag.

Returns: —
Triggers: —

changeColor
Open the color dialog and set the selected characters to the selected color.

Returns: —
Triggers: —

changeFont
Open the font dialog and set the selected characters to the selected font.

Returns: —
Triggers: —

changeParagraph
Open the paragraph format dialog to change the paragraph format.

Returns: —
Triggers: —

changeTabs
Open the change tab dialog to change the tab stop positions.

Returns: —
Triggers: —

clear
Deletes all of the text in the pane and clears the modified flag.

Returns: —
Triggers: changed:

clearModified
Clears the modified flag.

Returns: —
Triggers: —

clearSelection
Deletes the selected text from the pane and sets the modified flag.

Returns: —
Triggers: —

copySelection
Copies the currently selected text to the system clipboard.

Returns: —
Triggers: —
**Rich Edit (Win32 only)**

**cutSelection**
Copies the currently selected text to the system clipboard and then deletes it from the pane. The modified flag is set. This is the same as **copySelection** followed by **clearSelection**.

*Returns:* —
*Triggers:* —

**disable**
Disables the rich edit control so it does not respond to user input.

*Returns:* —
*Triggers:* —

**enable**
Enables the rich edit control to respond to user input.

*Returns:* —
*Triggers:* —

**growTextLimit**
Increase the text limit by another 32K bytes.

*Returns:* —
*Triggers:* —

**insert:** *aString*
Replaces the current selection with the argument. The argument is inserted at the current input point if there is no text selection. The modified flag is set.

*Returns:* —
*Triggers:* —

**isModified**
Triggers **modifiedIsTrue** if the modified flag is set and returns true. Returns false if the modified flag is not set. The modified flag is not changed.

*Returns:* a Boolean
*Triggers:* **modifiedIsTrue**

**pasteSelection**
Replaces the current selection with the text in the system clipboard. The clipboard text is inserted at the current input point if there is no text selection. The modified flag is set.

*Returns:* —
*Triggers:* —

**print**
Open a print dialog to print the contents on the selected printer.

*Returns:* —
*Triggers:* —

**saveToFile:** *aString*
Saves the contents of the rich edit control to the file whose path name is given by the argument. The modified flag is not affected.

*Returns:* —
*Triggers:* **fileException:**
selectAll

Selects all the text in the rich edit control.

Returns: —
Triggers: —

selectedText

Returns the currently selected text. An empty string is returned if no text is selected.

Returns: a String
Triggers: —

setFocus

Moves the input focus to the rich edit control.

Returns: —
Triggers: —

setFromFile: aString

Sets the contents of the rich edit control from the contents of the file whose path name is given by the argument. The modified flag is cleared.

Returns: —
Triggers: changed
fileException:

setModified

Sets the modified flag to true.

Returns: —
Triggers: —

setValue: aString

Sets the contents of the rich edit control to the argument and clears the modified flag. The argument can be either plain text or formatted text.

Returns: aString
Triggers: changed:

undo

Undo the last user edit in the text pane. Sets the modified flag and triggers the changed: event if the undo operation changes the text.

Returns: —
Triggers: —

unformattedContents

Returns the contents of the rich edit control as a plain text string (without formatting information).

Returns: a String
Triggers: —

value

Returns the formatted contents of the rich edit control.

Returns: a String
Triggers: —
Direct-Edit

Direct-editing the rich edit part opens a dialog which lets you enter and format the initial contents.

Property Edit

Each of the part-specific controls in the rich edit part's Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.

Edit contents...

Pressing this button opens the following dialog, which lets you enter and format the initial contents:
Drag drop...

Opens the following dialog where you can enable the capability for objects to be dragged from and/or dropped on the part:

The Source and the Target check boxes in the Drag group pane are used to enable (or disable) the part as the source and/or the target for a specified set of drag and drop operations.

The Unused and the Chosen and Prioritized list panes in the Drag Target Formats group pane are used to indicate how the part should render a dropped object, and they are disabled unless the part has been enabled as a target. This target part attempts to render the dropped object in one of the chosen formats, beginning with the first entry in the Chosen and Prioritized list pane. If the object cannot be rendered in that format, then the other chosen formats are tried. If the object cannot be rendered in any of the chosen formats, then it is ignored. To move formats between the Unused and the Chosen and Prioritized list panes, drag them from one list to the other. To change the priority order of the chosen formats, drag them within the Chosen and Prioritized list pane.

The Select Operations multiple-selection list pane in the Drag Target Operations group pane is used to indicate the set of drag and drop operations that the part will accept, and it is disabled unless the part has been enabled as a target. An operation is selected if it is highlighted.

The Defaults button is used to restore the part's drag drop property to its default values.
Rich Edit (Win32 only)

Border
If checked, a border appears around the pane. The default is to display the border.

Read only
If checked, the contents of the pane cannot be edited at runtime. The default is to allow editing of the contents.

Related Parts
- **Format Menu (Win32 only)**
  The format menu is a visual part that implements a prefabricated menu for operations on text using the rich edit part.

Example
The rich edit part is used in the EDITOR.PAR sample application, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. Operations within the rich edit control are linked to commands on the File, Edit, and Format menus. Some of these commands are also available as toolbar buttons.

For additional explanation of:
- This sample application, see the example in the Toolbar (Win32 only) chapter later in this document
- The format menu part used in this sample application, see the example in the Format Menu (Win32 only) chapter earlier in this document
- Using menus, see the Menu chapter earlier in this document
Scroll Bars
(Win32 only)

A scroll bar, either vertical or horizontal, is a visual part that allows you to set an integer value by moving the scroll box along the scroll bar.

You define a scroll bar’s range of values and its initial value by setting its minimum, maximum, and initial values in its Properties dialog.

You can use a scroll bar to allow a user to visually set values that have familiar increments, such as feet, inches, and degrees. They can also be used when immediate visual feedback is necessary, such as an application that allows users to blend colors. The two types of scroll bars are shown in the following illustration:

![Scroll Bars Illustration]

Operation

You can operate the scroll bar in the following ways:

- Drag the scroll box
- Click on the scroll bar itself (on either side of the scroll box) to move the scroll box 24 increments in the direction of the click
- Click on one of the arrow buttons at either end of the scroll bar to move the scroll box one increment in the direction indicated by the arrow
- Send the `setValue:` message to the scroll bar

Events

`changed: anInteger`

Occurs whenever a new value is set. The event value is the new scroll bar value.

Triggered By: the user

`setValue:`

`rightClicked`

Occurs when the user clicks mouse button 2 anywhere on the scroll bar.

Triggered By: the user
Scroll Bars (Win32 only)

**tabbed**
Occurs when the scroll bar has input focus and the user presses the Tab key. You can connect this event to the `setFocus` message of another part to set the tab order.

*Triggered By: the user*

**Messages**

**disable**
Sets the scroll bar to ignore user input. The scroll bar can still be set with the `setValue:` message. The arrows of a disabled scroll bar are dimmed and the scroll box is missing.

*Returns: —*

*Triggers: —*

**enable**
Sets the scroll bar to accept user input.

*Returns: —*

*Triggers: —*

**setFocus**
Moves the input focus to the scroll bar.

*Returns: —*

*Triggers: —*

**setValue: anInteger**
Sets the scroll bar's value to `anInteger`. If the new value is not within the scroll bar's range, the scroll bar's value is not changed and the `changed:` event is not triggered.

*Returns: —*

*Triggers: `changed:`*

**value**
Returns the current scroll bar value.

*Returns: —*

*Triggers: —*

**Direct-Edit**

There is no direct edit for a scroll bar.
Property Edit

Each of the part-specific controls in the scroll bar’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

### Initial value

This controls the value of the scroll bar and the position of the scroll box when the application is first opened.

### Minimum

The minimum value of the scroll bar’s range.

### Maximum

The maximum value of the scroll bar’s range.

**Related Parts**

- **Dial Pane**
  
  The dial pane is a circular gauge that has capabilities similar to the scroll bar as well as a number of options for displaying its current setting.
Example

The example demonstrates the operations of a scroll bar. Dragging the scroll box with the mouse triggers the scroll box's `changed:` event. The changed event is linked to the `setvalue:` messages of both the dial pane and the entry field. Any change in the scroll bar is reflected in both places. Similarly changing either the dial pane or the entry field updates the position of the scroll box in the scroll bar.

The disable button sends the disable message to the scroll bar, disabling it for direct user input. The scroll bar can still be set, however, by using the dial pane or the entry field. The enable button enables direct user input to the scroll bar. The set focus button shifts input focus to the scroll bar. When the scroll bar has focus, the scroll box can be set using the arrow keys.

This example is in the file SCROLLBA.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Sliders
(OS/2 only)

A slider control is a visual part that lets a user set, display, or modify a value by moving the slider arm along the slider shaft. The slider arm shows the value that is currently selected by its position on the slider shaft. Slider values are set by changing the location of the slider arm.

There are two sliders in the Parts Catalog—a horizontal slider and a vertical slider. Examples of these sliders are shown in the following illustration.

A slider has a linear scale and a range with divisions indicated by tick marks. The number of tick marks, their length, and their labeling can be set in the Properties dialog. A slider has an integer value, which is between zero and the number of tick marks minus one.

Sliders are typically used to allow a user to visually set values that have familiar increments, such as feet, inches, degrees, decibels, and so forth. However, they can also be used for other purposes when immediate visible feedback is necessary, such as to blend colors or to show the percentage of a task that has completed. For example, an application might allow a user to mix and match color shades by moving a slider arm, or a read-only slider could be provided that shows how much of a task has completed by filling in the slider shaft as the task progresses.

The slider is designed to be customizable to meet varying application requirements, while providing an easy-to-use interface part. The application can specify different scales, sizes, and orientations for its sliders.

A black square is drawn in the center of the slider to show that the slider control window has the input focus. The arm indicates the value. An arm lets you change values. A slider control can be used without the arm (a read-only slider) to display values but not permit them to be changed. The ribbon strip fills the shaft between the zero tick mark and the arm.
A slider also has arrow buttons which allow you to change its value and the position of the arm.

Operation

The slider value can be changed in the following ways:

- Dragging the slider arm with mouse button 1 or 2 causes it to move in the direction that the cursor is moving. When the mouse button is released, the value closest to the slider arm's position becomes the value selected.
- Clicking on the slider shaft with mouse button 1 causes the slider arm to move one increment in the direction of the cursor.
- Clicking on the slider shaft with mouse button 2 causes the slider arm to move to the cursor's location.
- Two arrow buttons are available. The arrows on the arrow buttons point to opposite ends of the slider. Both arrow buttons are positioned at the same end of the slider. Clicking either of the arrow buttons with mouse button 1 moves the arm one increment in the appropriate direction.
- A new value can be provided as an argument with the setValue: message.

Events

changed: anInteger
Occurs whenever the value of the slider is changed. The new slider value is provided by the event value.

Triggered By: the user
setValue:

rightClicked
Occurs when the right mouse button is clicked when the cursor is over the slider.

Triggered By: the user
tabbed
Occurs when the Tab key is pressed and the slider has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

Messages

disable
Ignores user input.

Returns: —
Triggers: —

enable
Enables the slider to respond to user input.

Returns: —
Triggers: —
setFocus
Moves the input focus to the slider.

Returns:  
Triggers:  

setValue: anInteger
Sets the slider value to the argument. If the value is in the range of zero to the maximum increment, the new value is set and the changed: event is triggered. Otherwise, no value is set and no events are triggered.

Returns:  
Triggers:  changed:

value
Returns the current slider value.

Returns:  an Integer  
Triggers:  

Direct-Edit
Direct-editing a slider lets you drag the arm to set the initial value.

Property Edit
The property edit dialog for a horizontal slider part is shown below. (For a vertical slider, the Scale and Scroll button groups are reversed in position). Each of the part-specific controls in the slider's Properties dialog is described in the following paragraphs.

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Styles

• **Scale**
  Depending on vertical or horizontal slider, the scale (tick marks and labels) can be either located above or to the right of the slider shaft, or below or to the left of the slider shaft. The default location is above or to the right.

• **Scroll Button**
  Depending on vertical or horizontal slider, the two scroll buttons can be either located above or to the right of the slider shaft, or below or to the left of the slider shaft. The default location is to the right or above.

• **Ribbon strip**
  As the slider arm moves, the slider fills the slider shaft between the home position and the slider arm with a color value that is different from the slider shaft color, similar to the mercury in a thermometer. The ribbon strip does not appear by default.

• **Snap to increment**
  The slider arm, when moved to a position between two specified values on the slider scale, such as between two tick marks, is positioned on the nearest value and is redrawn at that position. If this style is not specified, the slider arm remains at the position to which it is moved. However the value of the slider is always the value of the nearest increment. Intermediate values cannot be obtained by moving the slider arm between tick marks. By default, the slider arm does not snap to increments.

• **Read only**
  This means that the user cannot interact with the slider. It is used merely as a mechanism to present a quantity, such as the percentage of completion of an ongoing task. Visual differences for a read-only slider are:
  - a narrow slider arm
  - no knob
  - no slider buttons

**Edit contents...**

Pressing this button brings up the slider’s Edit dialog, as shown in the following illustration. This dialog lets you set the number of tick marks (which determines the range of values), the spacing between the tick marks (which determines the overall length of the slider), the default length for tick marks, the initial value, and default and custom labeling for the tick marks.
• **Number of ticks**
  The value in this field determines the number of tick marks on the scale and the range of integer values the slider assumes. The range is from zero to the number of increments less one. For example, the default horizontal slider has 81 increments, and can have values between zero and 80.

• **Spacing between ticks**
  This value indicates the distance (in pixels) between two adjacent tick marks. This setting controls the overall length of the slider.

• **Default tick length**
  This value sets the length of unmodified tick marks in pixels. If the ticks length is set to zero, the tick marks are invisible.

• **Initial setting**
  You can enter an initial value for the slider in this field.

• **Modify ticks by interval**
  Lets you specify that the ticks the specified number of units apart have a different length. If Automatic labels is checked, then the modified ticks are labeled with their value. The default is to not modify ticks by interval.

• **Modify individual ticks**
  Lets you specify a length and label for individual tick marks.

**To modify an individual tick mark:**

1. Enter the number of the tick to be modified in the Tick spin button. The tick number is the same as the tick value, so zero is the first tick mark.
2. Enter the length in pixels for this tick mark into the Length field.
3 Enter the label for this tick mark in the Label field.
4 Repeat steps 1 through 3 for any additional tick marks you want to modify.

- **Clear modified ticks**
  Erases all tick modifications. You should use this before setting up your own tick modifications to clear out the tick modifications used by the default sliders.

- **Clear tick labels**
  Erases the labels from all modified tick marks, while leaving any length changes.

**Related Parts**

- **Dial Pane**
  The dial pane part is a circular gauge that behaves much like the slider.

**Remarks**

You will generally want to use a ribbon strip on a read-only slider.

**Example**

The example contains two sliders. The bottom one is read-only and has the ribbon strip on. Operating the first slider or entering a number in the first field changes the second slider to that value and displays the value in digits in the second field.

This example is in the file SLIDER.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The speaker is a nonvisual part that gives you access to the system speaker.

You can generate predefined sounds with the speaker's **tone**, **alarm**, or **ring** messages, or you can program your own sounds.

The speaker part has two properties, frequency and duration, which apply only to the **tone** message. These properties control the pitch of the sound and the length of time it persists. These properties can be changed in the Properties dialog or with the **setFrequency**: and **setDuration**: messages.

### Events

The speaker part has no events.

### Messages

**alarm**

Makes an alarm sound.

*Returns:* —  
*Triggers:* —

**frequency**: `frequencyInteger`  
**duration**: `durationInteger`

Makes a sound having the frequency `frequencyInteger` and the duration `durationInteger`. This message does not change the frequency and duration properties.

*Returns:* —  
*Triggers:* —

**ring**

Makes a ringing sound by alternating the frequency between 700 and 900 Hertz every 100 milliseconds.

*Returns:* —  
*Triggers:* —

**setDuration**: `anInteger`

Changes the duration property that controls the length, in milliseconds, of the sound made by the **tone** message.

*Returns:* —  
*Triggers:* —

**setFrequency**: `anInteger`

Sets the frequency property that controls the frequency, in Hertz, of the sound made by the **tone** message.

*Returns:* —  
*Triggers:* —
Speaker

**tone**

Makes a sound using the current frequency and duration property values.

- **Returns:** —
- **Triggers:** —

**Direct-Edit**

Direct-editing the speaker part changes the part name that appears beneath the icon in the workbench.

**Property Edit**

Each of the part-specific controls in the speaker’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

![Speaker Properties Dialog](image)

**Frequency**

The frequency property sets the pitch, in Hertz, of the tone sounded by the `tone` message. The frequency is set to 800 Hertz by default.

**Duration**

The duration property sets the length of time, in milliseconds, the note sounded by the `tone` message lasts. The duration is set to 200 milliseconds by default.

**Remarks**

The `tone` and `frequency:duration:` messages won’t make any sound if the frequency or duration are set to zero. The sound will be inaudible if the frequency is set to values outside the range of 40 to 10000 Hertz or if the duration is less than 20 milliseconds (approximately).
This example is in the file SPKRXMLP.LPAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Spin Buttons
(OS/2 only)

A spin button is a visual part that lets you make selections from a fixed range of
values; for example, a day of the week or a month of the year.

A spin button has a text entry field with up-arrow and down-arrow scroll buttons on
its right. Pressing the scroll buttons causes each of the possible values to be
displayed in sequence in the entry field. The **changed:** event is triggered when you
release the scroll button. You can also type the value into the spin button field from
the keyboard.

Each field has a type that is either numeric or list of text items. The type and range
of values that can be selected for each type are set in the property dialog or with
direct edit.

You can control several spin buttons with a single set of scroll buttons. There are
two spin button parts in the Catalog. Master spin buttons have scroll buttons. Slave
spin buttons do not have scroll buttons and must be associated with a master spin
button through their property dialog to be operated. The result of using master and
slave spin buttons together is that there are several spin button entry fields sharing a
single set of scroll buttons. The spin button entry field that has the input focus is the
one that is changed by the arrow buttons.

The following illustration shows three spin buttons used to set the date. The two
spin buttons on the left are slaves to the master spin button on the right:

![Spin Buttons Illustration]

Master and slave spin buttons share the same events and messages and set their type
and range of values in the same way.

**Events**

- **changed:** *aString or anInteger*

  Occurs when either the up or down scroll arrow is released. The event value
  is the new spin button value – an integer for a numeric spin button or a
  string for a text spin button.

  Triggered By: the user

- **rightClicked**

  Occurs when mouse button 2 is clicked over the spin button.

  Triggered By: the user
Spin Buttons (OS/2 only)

spinnedAround: anInteger
Occurs when the spin button has spun around its entire range. anInteger is 1 if the spinning direction is up and -1 if the spinning direction is down.

Triggered By: the user

tabbed
Occurs when the Tab key is pressed and the spin button has the input focus. You can connect this event to the setFocus message of another part to set the tab order.

Triggered By: the user

Messages

disable
Returns: —
Triggers: —
Deactivates the spin button so it does not respond to user input.

enable
Enables the spin button to respond to user input.
Returns: —
Triggers: —

setFocus
Moves the input focus to the spin button.
Returns: —
Triggers: —

setList: aList
If the spin button is numeric, set the minimum and maximum values to aList where aList is a two element array containing the minimum and maximum values respectively of the numeric range. If the spin button uses a list of items, set the items to aList, where aList is a collection of strings.
Returns: —
Triggers: —

setValue: aString or anInteger
Sets the selection to the argument. If the argument is an integer, the item having that index is selected. If the argument is a string, the selection is set to the first item that matches the string. If the argument is outside of the range set for a numeric spin button or not in the item list for a text spin button, the selection is not changed.
Returns: aString or anInteger
Triggers: —

spin: anInteger
Spin the button field anInteger number of times. Spins up if anInteger is positive, down if anInteger is negative.
Triggers: —
Spin Buttons (OS/2 only)

value: aString or anInteger
Returns the current selection – an integer for numeric spin buttons; a string for text spin buttons.

Returns: aString or anInteger
Triggers: —

Direct-Edit

Direct editing either a master or a slave spin button opens the Edit subdialog of the Properties dialog which is described below.

Property Edit for Master Spin Buttons

Each of the part-specific controls in the master spin button’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Styles

- **Read only**
  Text cannot be entered into a read-only spin button. Text can be entered by default.

- **No border**
  The spin button is displayed without a border when this is checked. The default is to display the border.

- **Fast spin**
  When checked, increases the speed the selections are shown when a button is pressed. The default is to not have fast spin enabled.

Visual Smalltalk Enterprise Part Reference 455
Spin Buttons (OS/2 only)

- **Text align**
  The text in the field can be left justified, right justified, or centered. The default is right justified.

**Edit...**

Opens the Edit subdialog which lets you select the type of spin button (numeric range or list) and set the range of values that can be selected.

The Edit subdialog has Numeric range selected for the type of spin button, so the entry fields for the minimum and maximum values are enabled. These values determine the lower and upper limits of the range of numbers that the numeric spin button will scroll through.
If you change the type to List, the text pane at the bottom is enabled so you can enter a list of text items for the range of values that the list spin button will present.

Enter the items in the order you want the list spin button to present them.
Property Edit for Slave Spin Buttons

The Properties dialog for a slave spin button is the same as that of the master spin button except for the addition of the Set master... button. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Set master...

Pressing the Set master... button opens the Set Spin Button Master dialog.

Select one of the master spin buttons in your application to be the master spin button for this slave spin button and press OK.
Remarks

The type of spin button and range of values cannot be changed at application runtime.

Example

The example contains both numeric and list spin buttons, as well as master and slave spin buttons. When you operate the spin buttons, the values provided by the changed events of the three spin buttons are displayed in static text parts below the spin buttons. The buttons on the right let you experiment with disabling, enabling, and setting focus to the middle spin button.

This example is in the file SPINBTN.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Spin Buttons (OS/2 only)
A static graphic is a visual part that displays a referenced graphic. In Windows, the graphic can only be an icon. In OS/2, the graphic can be either an icon or a bitmap.

The graphic initially displayed by a static graphic can be
- set from a file
- set from the clipboard
- entered with the Icon Editor (for a Windows or OS/2 icon)
- entered with the Bit Editor (for an OS/2 bitmap)

The graphic can be changed at runtime with the `setValue:` message.

**Events**

A static graphic part does not have any events.

**Messages**

`value`

Returns the contents of the static graphic.

*Returns:* an Icon (or a Bitmap, for OS/2)

*Triggers:* —

`setValue: anIcon (or aBitmap, for OS/2)`

Sets the contents of the static graphic to the argument.

*Returns:* anIcon (or aBitmap, for OS/2)

*Triggers:* —

**Direct-Edit**

Direct-editing opens the Icon Editor (for a Windows or OS/2 icon) or the Bit Editor (for an OS/2 bitmap).
Property Edit

Each of the part-specific controls in the static graphic’s Properties dialog is described in the following paragraphs. Note that the OS/2 version of this part has the additional property of Styles (bitmap or icon).

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Styles

For the OS/2 version of this part, you can choose the type of graphic to be displayed.

- **Bitmap**
  When selected, the static graphic can display bitmaps.

- **Icon**
  When selected, the static graphic can display icons.

Edit graphic...

Opens the Icon Editor (for a Windows or OS/2 icon) or the Bitmap Editor (for an OS/2 bitmap). Allows you to edit the graphic or read a new one from a file.
Static Text

A static text part is a visual part that provides a single line of text to label controls or convey information.

The contents of a static text can be changed at runtime with the `setValue:` message. When you direct-edit a static text part and enter a new text string, the width of the static text part automatically adjusts so you can see all the characters.

Static text part properties let you right-justify, left-justify, or center the text within the boundary of the part, although the borders are not visible at runtime.

A static text part has messages to disable and enable it. Disabling a static text part changes its appearance by dimming the displayed text.

The following illustration shows static text used to query the user:

![Static Text Example](image)

Events

`mnemonicTyped`

Occurs when the user presses the shortcut key defined for this static text.

*Triggered By:* the user

Messages

`disable`

Dims the text currently displayed by the static text part.

*Returns:* —

*Triggers:* —

`enable`

Restores the appearance of the text in a disabled static text part.

*Returns:* —

*Triggers:* —

`setValue: aString`

Sets the text displayed by the static text part to `aString`.

*Returns:* `aString`

*Triggers:* —
value

Returns the text in the static text part.

*Returns*: a String

*Triggers*: —

**Direct-Edit**

Direct-editing a static text part lets you enter the text to be displayed. The width of the static text part automatically adjusts to accommodate the entered text.

**Property Edit**

Each of the part-specific controls in the static text’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

![Static Text Properties Dialog](image)

**Label**

You can enter the text to be displayed in this field or by direct-editing.

**Alignment**

You can select whether the text displayed is left-aligned, right-aligned, or centered with respect to the left and right borders of the static text part. The default is left-aligned. The borders are not visible at runtime.

If the text is changed with the `setValue:` message, the size of the static text does not change. If the new text is too long to be displayed, it is truncated.

**Related Parts**

- **Group Pane**
  
  The group pane part is used to label groups of parts.
**Remarks**

You can define a shortcut key that will trigger the `mnemonicTyped` event when that key is pressed. You could use this event, for example, to set the focus to a related control or to sound the speaker. To define a shortcut key in Windows, insert the ampersand (&) just before the shortcut character in the static text. In OS/2, the tilde (~) character is used instead of the ampersand. The character following the ampersand or tilde becomes the shortcut key.

When the static text is displayed, the ampersand or tilde is not shown, and the shortcut character is underlined to indicate that it is the shortcut key for the static text. If the part that currently has the input focus expects keyboard input (like an entry field), you need to press and hold the Alt key and then press the shortcut key.

**Example**

This example lets you change the text displayed by a static text part at runtime. The new text is entered in the entry field. You can also disable and enable the static text to see how its appearance changes.

This example is in the file `STATICTE.PAR` in the `SAMPLE\PARTDEMO\STATICTE.PAR` subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Status Window (Win32 only)

A status window is a short horizontal control, usually appearing at the bottom of a parent window, where the application can display various kinds of status information. This control can include an optional size grip (displayed in its lower right corner) that makes it easier to resize the containing window.

The status window has two display modes: simple and nonsimple. In simple mode, the whole status window is used to display a single piece of text. In nonsimple mode, the window can be divided into multiple fields where each field contains its own text or graphics and has its own visual types. The two modes can be switched back and forth with each remembering its own contents.

The following illustration shows the Windows 95 Explorer with a status window having two fields:

Operation

When you first drag a status window into the workbench for your application, it is set in simple mode. When you create fields (using the Fields page of the Properties dialog), it is automatically switched into nonsimple mode unless the Simple mode (ignore fields) property (on the Simple Mode page of the Properties dialog) is checked.

Events

A status window has the following event.

rightClicked

Occurs when the user clicks somewhere on the status window with mouse button 2.

Triggered By: the user
Messages

For each status field created, two messages are automatically generated. The message for setting the text to be displayed in a field has a name composed of the field name prefixed by `set`. The message for retrieving the text from a field has the same name as the field.

In addition to these generated messages for the status fields, a status window has the following messages.

- **simpleMode**: `aBoolean`
  - Switches between the simple mode (when the argument is true) and the nonsimple mode (when the argument is false) of the status window.
  - **Returns**: —
  - **Triggers**: —

- **simpleModeText**: `aString`
  - Sets the simple mode text to the argument.
  - **Returns**: —
  - **Triggers**: —

Direct-Edit

Direct-editing the status window allows you to enter the initial text for the simple mode.

Property Edit

Each of the part-specific controls in the status window's Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.
Place At Top
If checked, the status window is placed at the top of its containing window. The default is to place the status window at the bottom of its containing window.

Size Grip
If checked, the status window includes an optional size grip (displayed in its lower right corner) that makes it easier to resize the containing window. The default is to not include this size grip.
**Fields**
This combo box is used to add, delete, or modify status fields in the status window.

**To add a new status field:**
1. Enter a unique name in the text entry field.
2. Select its Field Type.
3. Enter its Contents.
4. Set its Width.
5. Click on the Add button.

**To delete an existing status field:**
1. Select the field from the list.
2. Click on the Remove Field button.

**To modify an existing status field:**
1. Select the field from the list.
2. Modify the field's Contents, Width, or Field Type.
3. Click on the OK button when you are finished.

**Field Type**
This group of radio buttons is used to select the type of the currently selected status field. Each type has a different appearance, as shown in the following illustration:

<table>
<thead>
<tr>
<th>first one</th>
<th>second one</th>
<th>last one</th>
</tr>
</thead>
</table>

- **caveIn:** the field appears “caved-in” (the default setting), as shown in the first field of the previous illustration.
- **popOut:** the field appears “popped-out”, as shown in the second field of the previous illustration.
- **noBorder:** the field appears without a border, as shown in the last field of the previous illustration.
- **ownerDraw:** the field will be drawn by the user.

**Contents**
This text entry field is used to set the initial contents of the currently selected status field. Subsequent updates are made with the generated `set` message for this status field.

**Width**
This integer entry field is used to set the width (in pixels) of the currently selected status field. Text that is longer than the width of the field is truncated during display.
Minimum Height
This integer entry field is used to set the minimum height (in pixels) of the status window.

Simple mode
If checked, the status window is initially opened in simple mode even if status fields are defined. If unchecked and status fields are defined, the status window is initially opened in nonsimple mode. The default is to open the status window in nonsimple mode if status fields are defined.

Simple mode text
This text entry field is used to enter the initial text for a status window in the simple mode. Subsequent updates are made with the `simpleModeText:` message.
Example

The status window is used in the EDITOR.PAR sample application, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. The status window at the bottom of the application is used to display the tip text for the toolbar buttons, along with timing information.

For further explanation of this sample application, see the example in the Toolbar (Win32 only) chapter later in this document.
String Holder

A string holder is a nonvisual part that serves as a variable to hold a String.

The string holder is a value holder with the class set to String so that additional
String-specific messages are available. A string holder can be used as a constant or as
an initialized variable by setting the initial value in the Properties dialog.

The value can be accessed at runtime with the `value` message and can be set with
the `setValue:` message. The `changed:` event is triggered by the `setValue:` message.
See the `Value Holder` chapter later in this document for information about events and
messages common to all value holders.

The initial value ("a string" by default) appears in the string holder on the
workbench. The following illustration shows several string holders on the
workbench:

Long strings are truncated when displayed. An ellipsis (...) is appended to the end of
the string to indicate that there is more. The full string is visible in the Properties
dialog and when you direct-edit the string holder.

Event

A string holder is a specialization of a value holder. In addition to the standard value
holder events, it has the following string holder specific event.

changed: aString

Occurs when a new string is appended. Provides the new string as an event
value.

Triggered By: `appendToValue:`
A string holder is a specialization of a value holder. In addition to the standard value holder messages, it has the following string-holder specific messages.

< aString
Returns true if the value of the string holder is less than aString, otherwise returns false.

Returns: a Boolean
Triggers: —

<= aString
Returns true if the value of the string holder is less than or equal to aString, otherwise returns false.

Returns: a Boolean
Triggers: —

> aString
Returns true if the value of the string holder is greater than aString, otherwise returns false.

Returns: a Boolean
Triggers: —

>= aString
Returns true if the value of the string holder is greater than or equal to aString, otherwise returns false.

Returns: a Boolean
Triggers: —

appendToValue: aString
Set the value to the result of appending aString to the current value. This message modifies the string value in the string holder. This is in contrast to the concatenate: aString message, which returns the result of appending aString to the current value but does not change the value of the string holder.

Returns: —
Triggers: —

asArray
Returns the contents of the string holder translated into an Array. You can place the result into an array holder part by using a result link to the array holder's setValue: message.

Returns: an Array
Triggers: —

asArrayOfSubstrings
Divides the contents of the string holder into substrings at each occurrence of one or more space characters, returning an array of Strings.

Returns: an Array of Strings
Triggers: —
String Holder

asAsciiZ
Returns a new string containing all the characters in the string holder followed by the ASCII null (zero) character.

Returns: a String
Triggers: —

asDate
Returns the contents of the string holder converted into a value of type date. Valid formats for Strings that can be converted to dates are:
- Dec 31, 1989
- 31 Dec 1989
- 12/31/89
You can use a three-letter abbreviation for the month name, or you can spell it out. You can use a four-digit year or just the century digits. The delimiters between the month, day and year can be any sequence of non-alphanumeric characters.

Returns: a Date
Triggers: —

asFloat
Returns the contents of the string holder converted into a value of type Float. The contents of the string holder is expected to be a sequence of digits with an optional leading minus sign and optional signed exponent.

Returns: a Float
Triggers: —

asInteger
Returns the contents of the string holder converted into a value of type Integer. The contents of the string holder is expected to be a sequence of digits with an optional leading minus sign.

Returns: an Integer
Triggers: —

asLowerCase
Returns the contents of the string holder with all alphabetic characters in lower case.

Returns: a String
Triggers: —

asOrderedCollection
Returns the contents of the string holder translated into an Ordered Collection. You can place the result into an ordered collection holder by using a result link to the ordered collection’s setValue: message.

Returns: an Ordered Collection
Triggers: —
String Holder

asParameter
Returns the contents of the string holder translated into a form suitable for passing as a parameter to a DLL procedure to access host operating system services. For more information on this message see the DLL Accessor chapter earlier in this document.

Returns: —
Triggers: —

asSet
Translates the string holder into a member of class Set, eliminating any duplicate elements. You can create a part to hold a Set by defining the class of a value holder to be Set.

Returns: a Set
Triggers: —

asSortedCollection
Translates the string holder into a member of class SortedCollection containing the elements of the string holder sorted in ascending order. You can create a part to hold a SortedCollection by defining the class of a value holder to be SortedCollection.

Returns: a Sorted Collection
Triggers: —

asString
Returns the contents of the string holder as a String.

Returns: a String
Triggers: —

asSymbol
Returns a symbol whose characters are the same as the contents of the string holder. You can create a part to hold a Symbol by defining the class of a value holder to be Symbol.

Returns: a Symbol
Triggers: —

asUpperCase
Returns a String containing the contents of the string holder with alphabetic characters in upper case.

Returns: a String
Triggers: —

at: anInteger
The value stored at index anInteger.

Returns: a Value
Triggers: indexException:

at: anInteger put: aCharacter
Stores aCharacter at index anInteger. The argument must be of class Character, a single letter string won’t work. You can also create a character by using the string Holder’s asCharacter message.

Returns: —
Triggers: indexException:
concatenate: aCollection
Creates a new String containing the original followed by aCollection.
Collections other than Strings (such as the contents of a dictionary holder) are converted to Strings before concatenation.

Returns: a String
Triggers: —

copyFrom: beginning to: end
Creates a new String containing the contents of the string holder from index position beginning through index position end.

Returns: a String
Triggers: —

copyReplaceFrom: beginning to: end with: aCollection
Creates a new String with the elements at index positions beginning through index position end replaced with the elements of aCollection. aCollection can be any type of Collection, not just a String.

Returns: a String
Triggers: —

first
Returns the first element of the String.

Returns: a Value
Triggers: —

includes: aValue
Returns true if the string holder contains an element equal to aValue, otherwise returns false.

Returns: a Boolean
Triggers: —

isEmpty
Returns true if the size of the string holder is zero, otherwise returns false.

Returns: a Boolean
Triggers: —

last
Returns the last element of the string holder.

Returns: a Value
Triggers: —

occurrencesOf: aValue
Returns the number of elements in the string holder that are equal to aValue.

Returns: an Integer
Triggers: —

replaceControlChars
Replaces any control characters in the string holder with blanks. This operation does not apply to symbols.

Returns: —
Triggers: —
String Holder

reversed
Creates a new String containing the elements of the original in reverse order.

Returns: a String
Triggers: —

setFromReference: aReference
Retrieves the null-terminated string referenced by aReference. You get aReference as the result of a DLL accessor call message which returns a reference to a null-terminated string.

Returns: a String
Triggers: —

size
It returns the number of elements in the string holder.

Returns: an Integer
Triggers: —

trimBlanks
Returns a String that is the contents of the string holder with leading and trailing white space characters (spaces, tabs, line feeds, and carriage returns) removed.

Direct-Edit

Direct-editing a string holder lets you enter a new value. An ellipsis (...) appended to the end of a string indicates that it has been truncated for display. When you direct edit such a string holder, a dialog displaying the existing contents is displayed. You can make any desired changes in this dialog.

Property Edit

The part-specific control in the string holder's Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Value
You initialize a string holder by entering the initial string into this field. The default value is “a string”.

Remarks
The value is shown in the information area when the cursor is over the string holder.
String Holder
String Template

The string template is a nonvisual part that allows you to define a String with parameters.

The string template’s parameters are replaced with Strings at execution time. In the example below, the brackets indicate that fileName is a parameter which will be replaced by an actual filename at execution time.

Unable to open file [fileName].

Operation

A pair of messages is automatically generated for each parameter in the template. One message sets the value of the parameter, while the other message returns the current value of the parameter.

A parameter value is provided at application execution time by connecting a link to the set message for the parameter. Executing the value message returns the String in the string template, substituting the current value of each parameter in place of the bracketed parameter name.

A parameter name is a sequence of one or more alphanumeric characters surrounded by brackets. The Properties dialog allows you to define the type of brackets that the string template will interpret as parameter delimiters. If the literal String in your string template (the part of the String that you want displayed as it is typed) contains square bracket characters, you could use curly braces as the parameter delimiters. For example:

File access problem [Unable to open file {fileName}].

Events

There are no events for a string template.

Messages

setparameter: aString

Sets the value of the parameter named parameter to aString.

Returns: aString

Triggers: —

setTemplate: aString

Sets the template to aString.

Returns: aString

Triggers: —
String Template

**template**

Returns the String with parameter references.

*Returns:* a String

*Triggers:* —

**value**

Returns a String constructed by replacing the parameter references within the String with the current value of each parameter.

*Returns:* a String

*Triggers:* —

**Direct-Edit**

Direct-editing the string template part changes the part name that appears beneath the icon in the workbench.

**Property Edit**

Each of the part-specific controls in the string template’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

**Parameter markers**

Select the bracket character pair that will mark the parameter reference:

- Square brackets `[ ]`
- Curly braces `{ }`
- Parentheses `()`
- Angle brackets `< >`

Visual Smalltalk Enterprise Part Reference

482
Insert parameter reference

This button inserts a prototype parameter reference in the statement, replacing any selected text. The text within the brackets is the name of the parameter. You can change the name within the brackets to a name which describes the parameter you intend to provide.

Template

Enter the template string.

Remarks

The text within the parameter marker characters must be a single word consisting of alphanumeric characters and starting with a letter. This is the parameter name. You can have more than one reference to the same parameter in a template.

The parameter names that you use cannot be names already used as message names by the string template part. For example, you cannot use “template” as a parameter name, because it is already used as a message name. If name conflicts occur, you are warned, and must change the parameter.
String Template
Tab Control
(Win32 only)

A tab control is a visual part, similar to a notebook, that holds tab control pages having a labeled tab that can be selected to bring it to the front for viewing. This provides a compact and easily navigated means of displaying related groups of information, where each group appears on a separate page.

This part implements the Win95 tab control, which is commonly used to display the properties of an object. For an example of this type of usage, see the illustrations in the Property Edit topic later in this chapter.

Like any other visual part, the tab control itself must be contained in a window or dialog window. The tab control does not manage the behavior of any parts placed into its pages. It acts only as a container for the tab control pages.

A tab control page is a pane that contains other visual parts. A tab control page has only one level of tab, which can display either text or graphics and has properties that can be set. The input focus is indicated by a dashed line around the tab of the currently selected page.

The size and alignment of the tabs are controlled by various properties, which are fully explained in the Property Edit topic later in this chapter. To summarize:

- A tab's height is a fixed value specified by the Tab height property (on the Tab page).
- By default, a tab's width is variable and large enough to contain its text label and optional image. To use fixed-width tabs, check the Tab fixed width property (on the Style page) and specify a value for the Tab width property (on the Tab page).
- By default, scroll arrows are displayed when there is not enough room to display all the tabs. To have the tabs wrap onto additional lines, check the Multi line tabs property (on the Style page).

**Operation**

When you move or resize the tab control, the operation is automatically applied to all its tab control pages.

When you add a tab control page to a tab control, it is inserted after the current page. The page numbers are changed so that all pages are numbered in sequence. Tab control pages are identified by a number (starting with one) which is the sequential order in which it appears in the tab control.

Whenever the current page changes, the `changed:` and `changedPageNumber:` events are triggered. The `changed:` event provides the new page as the event value. The `changedPageNumber:` event provides the number of the new page as the event value.
The tab control pages can be turned by the user or by several messages. The messages `turnToPage:` and `turnToPageNumber:` select a page or page number. The messages `firstPage`, `lastPage`, `nextPage`, and `previousPage` access the pages sequentially.

**To turn the pages of a tab control:**

- Click on a tab.
- or
- Press the Page Up or Page Down keys (when the tab control has the input focus) to turn to the next or previous page.
- or
- Press the Home or End keys (when the tab control has the input focus) to turn to the first or last page.

To move between the pages of the tab control when you are building it, perform the following steps:

1. Double-click in the tab area of the tab control part to open its properties dialog.
2. If the Page page is not currently displayed, click on the Page tab.
3. Click on the desired page to select it, then click either the OK or the Apply button.

**Events**

A tab control has the following events.

**aboutToChange**

Occurs when a page is selected. If the `abortChange` message is executed as a result of this event, the page selection is canceled.

*Triggered By:* the user, `firstPage`, `lastPage`, `nextPage`, `previousPage`, `turnToPage:`, `turnToPageNumber:`

**changed:** `aPage`

Occurs when the tab control is turned to a different page. The value `aPage` is the new page.

*Triggered By:* the user, `firstPage`, `lastPage`, `nextPage`, `previousPage`, `turnToPage:`, `turnToPageNumber:`
changedPageNumber: anInteger
   Occurs when the tab control is turned to another page. The value anInteger is the page number of the newly selected page.
   Triggered By: the user
   firstPage
   lastPage
   nextPage
   previousPage
   turnToPage:
   turnToPageNumber:

inputOccurred
   Occurs when the user has entered input in one of the controls contained in the current page. (This event is often used to activate an “Apply” button that was initially disabled.)
   Triggered By: the user

rightClicked
   Occurs when the user clicks somewhere on the tab control (other than a tab control page) with mouse button 2. Each tab control page has its own rightClicked event.
   Triggered By: the user

tabbed
   Occurs when the Tab key is pressed and the tab control has the input focus. You can connect this event to the setFocus message of another part to set the tab order.
   Triggered By: the user

Messages

A tab control has the following messages.

abortChange
   Cancels a selection if executed as a consequence of triggering the aboutToChange event.
   Returns: —
   Triggers: —

disable
   Causes the tab control to ignore user selections.
   Returns: —
   Triggers: —

enable
   Enables the tab control to respond to user input.
   Returns: —
   Triggers: —
Tab Control (Win32 only)

**firstPage**  
Selects the first page of the tab control.  
*Returns:* —  
*Triggers:* aboutToChange  
*changed:*  
*changedPageNumber:*

**isFirstPage**  
Answers whether the current tab control page is the first one.  
*Returns:* a Boolean  
*Triggers:* —

**isLastPage**  
Answers whether the current tab control page is the last one.  
*Returns:* a Boolean  
*Triggers:* —

**lastPage**  
Selects the last page of the tab control.  
*Returns:* —  
*Triggers:* aboutToChange  
*changed:*  
*changedPageNumber:*

**nextPage**  
Turns to the page following the currently selected page.  
*Returns:* —  
*Triggers:* aboutToChange  
*changed:*  
*changedPageNumber:*

**pageNumber**  
Returns the page number of the currently selected page.  
*Returns:* an Integer  
*Triggers:* —

**previousPage**  
Turns to the page preceding the currently selected page.  
*Returns:* —  
*Triggers:* aboutToChange  
*changed:*  
*changedPageNumber:*

**setFocus**  
Moves the input focus to the tab control.  
*Returns:* —  
*Triggers:* —

**triggerApplyEvents**  
Triggers the **apply:** event for every page of the tab control.  
*Returns:* —  
*Triggers:* **apply:** (for every tab control page)
triggerHelpEvent

Triggers the *help:* event for the current page.

*Returns:*

---

*Triggers:*

- *help: (for the current tab control page)*

turnToPageNumber: anInteger

Changes the current page to the page whose page number is specified by *anInteger:*

*Returns:*

---

*Triggers:*

- *aboutToChange*
  - *changed:*
    - *changedPageNumber*

**Direct-Edit**

Direct-editing a tab control opens its Properties dialog.

**Property Edit**

Each of the part-specific controls in the tab control’s Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.
Ragged right
This property only takes effect when the Multi line tabs property is checked and the Tab fixed width property is not checked. When this property is checked (the default), the tabs in each row are ragged on the right. When this property is not checked, the tabs are padded with blanks so that the tabs in each row will be extended to reach the right edge of the tab control.

Tab as button
When checked, the tab appears as a simple push button. One example of this type of usage is the Win95 system task bar.

Tab tips
When checked, the tab control displays text describing the current page in a one-line pop-up window that appears beneath the cursor. This descriptive text is entered as the Tab tip property for a tab control page.

The default is to display this descriptive text.

Multi line tabs
When checked, the page tabs are allowed to wrap onto multiple lines if they cannot all fit within the tab control. The default is to have only one line of tabs, with scroll arrows appearing on the right if all the tabs do not fit.

Tab fixed width
When checked, the tabs on all pages are the same width (as specified in the Tab width property on the Tab page) and any text exceeding that width is truncated. The default is to have tabs of variable-width, which are long enough to contain the text and image for each page’s tab.
Tab Control (Win32 only)

Tab height
This integer entry field lets you set the height (in pixels) of the tab for all pages in the tab control.

Tab width
This integer entry field lets you set the width (in pixels) of the tab for all pages in the tab control. This property only takes effect when the Tab fixed width property (on the Style page) is set.

Demand load all pages
When this check box is selected, the tab control will create only the first page when it is launched, creating the rest of the pages as they are selected by the user. Check this box if you need to improve initial start-up time of your application.

The default is to create all the pages when the tab control is launched.
Select page

This list box allows you to easily select the current page (thus bringing it to the front for display) by clicking on it and then pressing the Apply or the OK button. You can also reorder a page in the tab control by dragging it to the desired position within the list.

There are initially no pages in the list. Pages appear in the list after tab control pages have been dragged from the catalog onto the tab control.

Related Parts

- Tab Control Page (Win32 only)
  The tab control page is a visual part which is used within a tab control (a container similar to a notebook), and has a labeled tab that can be selected to bring it to the front for viewing.

Example

This example (located in \SAMPLE\PARTDEMO\TABCTDEM.PAR) demonstrates the use of the Tab Control and Tab Control Page parts.

(All the illustrations in this section show a slightly modified version of the file, with the parts and links repositioned for greater clarity.)
The following illustration shows the links connecting the Tab Control to the other parts of the application.
This application provides online help, using the links shown in the following illustration.

This example is in the file TABCTDEM.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.

An additional example showing the use of the Tab Control part can be found in the Property Sheet (Win32 only) chapter earlier in this document.
A tab control page is a visual part which is used within a tab control (a container similar to a notebook), and has a labeled tab that can be selected to bring it to the front for viewing. This provides a compact and easily navigated means of displaying related groups of information, where each group appears on a separate page.

A tab control page is a pane that contains other visual parts. A tab control page has only one level of tab, which can display either text or graphics and has properties that can be set. The input focus is indicated by a dashed line around the tab of the currently selected page.

Tab control pages are commonly used to display the properties of an object. For an example of this type of usage, see the illustrations in the Property Edit topic later in this chapter.

### Operation

When you drop a tab control page on a tab control, it is placed after the current page in the tab control. The page numbers are changed so that all pages are numbered in sequence. Tab control pages are identified by a number (starting with one) which is the sequential order in which it appears in the tab control.

An empty page window is created when you create a new tab control page. You can place other visual parts in the page window. You can turn the tab control pages when you are editing an application in the workbench, as needed, to add parts and link them.

When you move or resize the tab control, the operation is automatically applied to all its tab control pages.

### Events

A tab control page has the following events.

**apply: aPage**

Occurs (for each page) when the triggerApplyEvents message is sent to the containing Tab Control.

* Triggered By: triggerApplyEvents (of the tab control)*

**back: aPage**

Occurs when the previousPage message is sent to the containing Tab Control. If this event is not connected to any action, then the previously viewed page is selected.

* Triggered By: previousPage*
deselected
Occurs when the user clicks mouse button 1 on a tab other than that of the current page, or when any of the tab control messages in the following list are used to turn to a different page.

Triggered By: the user
firstPage
lastPage
nextPage
previousPage
turnToPage:
turnToPageNumber:

help: aPage
Occurs (for the current page) when the triggerHelpEvent message is sent to the containing Tab Control. For example, the user pressed the Help button that you created for your tab control.

Triggered By: triggerHelpEvent (of the tab control)

next: aPage
Occurs when the nextPage message is sent to the containing Tab Control. If this event is not connected to any action, then the next physical page is selected.

Triggered By: nextPage

rightClicked
Occurs when the user clicks mouse button 2 on the current tab control page.

Triggered By: the user

selected
Occurs when the user clicks mouse button 1 on the current tab control page, or when any of the tab control messages in the following list are used to turn to this page.

Triggered By: the user
firstPage
lastPage
nextPage
previousPage
turnToPage:
turnToPageNumber:

Messages
A tab control page has no messages.

Direct-Edit
Direct-editing a tab control page opens its Properties dialog.
Property Edit

Each of the part-specific controls in the tab control page's Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to *Appendix C, Standard Part Properties*.

**Tab tip**

This text entry field allows you to enter the descriptive text that appears beneath the cursor when it lingers in the tab. For this descriptive text to be displayed, it must also be enabled in the containing tab control.

**Tab label**

This text entry field allows you to enter the text label that appears on the tab.

**Tab image**

This button opens the standard icon editor, where you can select an existing bitmap image or create a new one to be included on the tab.
Select page

This list box allows you to easily select the current page (thus bringing it to the front for display) by clicking on it and then pressing the Apply or the OK button. You can also reorder a page in the tab control by dragging it to the desired position within the list.

There are initially no pages in the list. Pages appear in the list after tab control pages have been dragged from the catalog onto the tab control.

Related Parts

- **Tab Control (Win32 only)**
  
  A tab control is a visual part, similar to a notebook, that holds tab control pages having a labeled tab that can be selected to bring it to the front for viewing.

  Most visual parts (except main windows) can be placed onto a tab control page.

Remarks

You can define a shortcut key for a textual page tab that will turn to a specific page when the tab control has the input focus and that key is pressed. To define a shortcut key, the character just before the shortcut character in the tab text must be the ampersand (&) character. The character following the ampersand becomes the shortcut key.

When the tab is displayed, the ampersand is not shown, and the shortcut character is underlined to indicate that it is the shortcut key for that page.
Example

This example demonstrates the use of the Tab Control and Tab Control Page parts. The previous illustration (of a slightly modified version of TABCTDEM.PAR, with the parts and links repositioned for greater clarity) shows the links connecting the Tab Control Page labeled 'C' to the other parts of the application.

This example is in the file TABCTDEM.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. For additional explanation of this example, see the example in the Tab Control (Win32 only) chapter earlier in this document.

An additional example showing the use of the Tab Control and Tab Control Page parts can be found in the Property Sheet (Win32 only) chapter earlier in this document.
Table Pane

The table pane is a visual part that resembles a spreadsheet.

The table pane can be used whenever tables, rows, or columns of data need to be displayed or modified. For example, a table pane can be used to display a calendar or to display database records. A table pane can be set to have a fixed or variable number of rows and columns.

There are two different ways to access and manipulate data in a table pane: indexed access and named access. These two different types of access offer various messages allowing you to specify cells, columns, rows, or matrices of cells. Named access allows you to access and manipulate data in the table pane using column and row names defined when the application is created. Indexed access allows you to access and manipulate data using row numbers and column numbers (beginning with 1 at the top left cell). You can combine both types of access in a single table pane. The possible combinations include:

- indexed columns, indexed rows
- indexed columns, named rows
- named columns, named rows
- named columns, indexed rows

Indexed access is based on the visual order of the columns—you must know the order of the rows and columns to access information properly. Named access is more flexible because the order of the rows and columns are no longer important.

Structurally the table pane is made up of either indexable collections (an array of arrays for example), nameable collections (a dictionary of dictionaries for example), or a combination of the two types.

If you are using nameable collections, such as a dictionary of dictionaries, the keys of the first dictionary correspond to the row names of table pane (which are set at edittime), while the keys of the contained dictionaries correspond to the column names of the table pane (which are set at edittime). If your table includes row or column names that don’t correspond to dictionary keys, the columns or rows are cleared. If the dictionary contains keys not corresponding to row or column names, the extra data is ignored.

Table panes also support access by unique cell index numbers. The cells in a 3 x 3 table pane would be indexed like this:

```
1 2 3
4 5 6
7 8 9
```
Table Pane

Table panes supply a cell’s index as an event value when you click or double-click on the cell at runtime.

You can assign default values to individual cells that will appear each time the application is run. Table pane properties allow you to specify whether or not labels and table contents can be changed by the user at runtime. You can also choose not to display column and/or row labels by setting a property. The contents of a cell is either a number or a String depending on the format (number or text) of the column or row. You choose the format in the Edit contents... subdialog of the Properties dialog. In cases where different row and column formats overlap, the row’s format takes precedence.

The following illustration shows a table pane used to record daily hours worked for several employees:

![Table Pane Example](image)

**Operation**

If user input has not been disabled by either the Read only property or the disable message, the user can select cells and edit them.

**To select cells:**

> Click on an individual cell.

This triggers the `selectedCell:`, `selectedIndex:`, and `selectedPoint:` events.

> Double-click on an individual cell.

This triggers the three `selected` events mentioned above, plus the `doubleClickedOnCell:` event.

> Drag over a range of cells.

This action selects all the cells in a rectangular region delimited by the cells where the drag was begun and terminated. All three of the `selected` events mentioned above are triggered when the mouse button is pressed, and `selectedGroup:` is triggered when the mouse button is released.

> Click on a row label or column label.

This selects the entire row or column of cells and triggers the `selectedGroup:` event.
To edit a cell:

> Select an individual cell and type the new contents.

When the modified cell is deselected, the `cellChanged:` event is triggered with the new contents of the cell as the event value.

If the Adjust Column Widths and Adjust Row Heights properties are set, the user can adjust the row height and column width.

To adjust the size of a column or row:

> Drag the dividing line between two column or row labels.

Events

**cellChanged: aValue**

Occurs when the contents of a cell is changed and the cell loses focus, or when the `setLabel:` message changes the contents for that cell. The event value is the new contents of the cell.

Triggered By: the user
clear
columnAt:put:
rowAt:put:
setLabel:\nsetName:
setValue:

**cellChangedAtRow: anInteger or aName column: anInteger or aName value: newValue**

Occurs when the table pane loses focus and its contents have changed. The row, column, and new value of the cell are passed as values. The event mechanism is designed to supply one value per event. In order to get more than one value from a single event, you must create a separate link for each value that you wish to use, with each link supplying one of the values. Each link provides a link label with a dot for each possible value but only one of the values can be used. In this case, you need to create three separate links from the `cellChangedAtRow:` event, or create a script that accepts three arguments (such as `#ProcessRow:Column:Value:`) which can all be obtained from a single link.

Triggered By: the user
clear
columnAt:put:
rowAt:put:
setLabel:
setName:
setValue:

**cellEnteredAtRow: anInteger or aName column: anInteger or aName value: aValue**

Occurs when the user presses the Enter key in a cell. The row and column of the cell are passed as arguments. The event mechanism is designed to supply one value per event. In order to get more than one value from a single event, you must create a separate link for each value that you wish to use, with each link supplying one of the values. Each link provides a link label with a dot for each possible value but only one of the values can be used. In this case, you need to create three separate links from the `cellEnteredAtRow:`
Table Pane

- **changed:** aMatrix
  Occurs when the table pane loses focus and the contents have changed.
  aMatrix is anArrayOfArraysOfValues, or anArrayOfNamableCollectionsOfValues, or aNamableCollectionOfArraysOfValues, or aNamableCollectionOfNamableCollectionsOfValues.

- **columnSelected:** anInteger or aString
  Occurs when the user selects an entire column by clicking in the column label. The column number or name is passed as the argument.

- **doubleClickedOnCell:** anInteger or aString
  Occurs when the user double-clicks on a cell. The value provided is the contents of the cell.

- **doubleClickedOnColumn:** anInteger or aString
  Occurs when the user double-clicks on a column label. The value provided is the column number or name.

- **doubleClickedOnRow:** anInteger or aString
  Occurs when the user double-clicks on a row label. The value provided is the row number or name.

- **rightClicked**
  Occurs when the user clicks mouse button 2 anywhere over the table.

- **rowEntered:** anInteger or aString
  Occurs when the user presses the Enter key in a row. The row number or name is passed as the argument.

- **rowSelected:** anInteger or aString
  Occurs when the user selects an entire row by clicking in the row label. The row number or name is passed as the argument.

- **selectedCell:** aValue
  Occurs when the user clicks or double-clicks on a cell. The value provided is the contents of the cell.
**selectedGroup: aMatrix**
Occurs when the user releases button 1. The value is the contents of the selected cells. *aMatrix* is anArrayof-ArraysOfValues, or anArrayOfNamableCollectionsOfValues, or anNamableCollectionOfArraysOfValues, or anNamableCollectionOfNamableCollectionsOfValues.

Triggers By: the user

**selectedIndex: anInteger**
Occurs when the user clicks or double-clicks on a cell. The value is the unique cell index.

Triggers By: the user

**selectedPoint: aPoint**
Occurs when the user clicks or double-clicks on a cell. The value provided identifies the row number of the cell with the Y coordinate of the point and column number of the cell with the X coordinate of the point.

Triggers By: the user

**tabbed**
Occurs when the Tab key is pressed and the table pane has the input focus. You can connect this event to the `setFocus` message of another part to set the tab order.

Triggers By: the user

**Messages**

**label**
Returns the value in the cell in indexed column *label* and the current row. Labels support “named” access for indexable collections. A row or column with indexed access selected generates label messages while a row or column with named access generates name messages. The table pane ignores labels assigned to named rows or columns and ignores names assigned to indexed rows or columns. An automatically generated label message is indistinguishable from an automatically generated name message.

Returns: a Value
Triggers: —

**name**
Returns the value in the cell in named column *name* and the current row.

Returns: a Value
Triggers: —
**Table Pane**

- **cellAtName: aString**
  Returns the value in the cell named by the argument using the format `RnCn` (the letter `R` followed by the row number followed by the letter `C` followed by the column number) or `An` (the column letter followed by the row number). Returns nil if there is no value at that location.
  
  Returns: a Value
  Triggers: —

- **cellAtPoint: aPoint**
  Returns the value in the cell whose column and row are given by the X and Y coordinates of the argument, or nil if there is no value at that location.
  
  Returns: a Value
  Triggers: —

- **cellAtRow: anInteger or aString column: anInteger or aString**
  Returns the value of the cell at the specified row and column.
  
  Returns: a Value
  Triggers: —

- **cellAtRow: anInteger or aString column: anInteger or aString put: aValue**
  Sets the cell at the specified row and column to `aValue`.
  
  Returns: aValue
  Triggers: —

- **clear**
  Clears the table pane.
  
  Returns: —
  Triggers: —

- **columnAt: anInteger or aString**
  Returns the values in the specified column. The leftmost column in the table is column one. `aCollection` can be either an indexable collection or a nameable collection depending on the access type.
  
  Returns: a Collection
  Triggers: —

- **columnAt: anInteger or aString put: aCollection**
  Sets the contents of the specified column to `aCollection`.
  
  Returns: —
  Triggers: selectedCell: selectedIndex selectedPoint

- **currentRowBottom**
  Sets the row cursor used by `label` and `setLabel` to the row with the last entry in the table pane.
  
  Returns: —
  Triggers: —
currentRowDown
Moves the row cursor used by label and setLabel down one row.

Returns: —
Triggers: —

currentRowTop
Sets the row cursor used by label and setLabel to the first row in the table pane.

Returns: —
Triggers: —

currentRowUp
Moves the row cursor used by label and setLabel up one row.

Returns: —
Triggers: —

disable
Causes the table pane to ignore user input.

Returns: —
Triggers: —

enable
Enables the table pane to respond to user input.

Returns: —
Triggers: —

matrixIn: aRectangle
Returns the values in the cells identified by aRectangle.

Returns: an Array of Arrays of Values
Triggers: —

rowAt: anInteger or aString
Returns the items in the specified row. a Collection can be either an indexable collection or a nameable collection depending on the row access type.

Returns: a Collection
Triggers: —

rowAt: anInteger or aString put: aCollection
Sets the contents of the specified row to aCollection. aCollection can be either an indexable collection or a nameable collection depending on the access type.

Returns: —
Triggers: selectedCell:
selectedIndex
selectedPoint

selectedCell
Returns the contents of the currently selected cell. Returns nil if no cell is selected.

Returns: a Value
Triggers: —
selectedGroup
Returns the contents of the currently selected group of cells. Returns nil if no group of cells is selected. aMatrix is anArrayOfArraysOfValues, or anArrayOfNamableCollectionsOfValues, or aNamableCollectionOfArraysOfValues, or aNamableCollectionOfNamableCollectionsOfValues.

Returns: a Matrix
Triggers: —

setLabel: aCollection or aValue
If the argument is a collection (e.g. an Array) other than a string, setLabel: sets the entire indexed column Label to the collection. If the argument is a number or a string, setLabel: sets the cell at the current row at the column Label to the number or string. Labels support “named” access for indexable collections. A row or column with indexed access selected generates label messages while a row or column with named access generates name messages. The table pane ignores labels assigned to named rows or columns and ignores names assigned to indexed rows or columns. An automatically generated label message is indistinguishable from an automatically generated name message.

Returns: aCollection or aValue
Triggers: cellChanged:

setName: aCollection or aValue
If the argument is a collection (e.g. an Array) other than a string, setName: sets the entire named column Name to the collection. If the argument is a number or a string, setName: sets the cell at the current row at column Name to the number or string.

Returns: aCollection or aValue
Triggers: cellChanged:

setColumnLabels: anArrayOfStrings
Sets the column labels to anArrayOfStrings. The column labels can also be set through the Properties dialog. Labels support “named” access for indexable collections. A row or column with indexed access selected will generate label messages while a row or column with named access will generate name messages. The table pane will ignore labels assigned to named rows or columns and names assigned to indexed rows or columns. An automatically generated label message is indistinguishable from an automatically generated name message.

Returns: —
Triggers: —

setColumnNames: anArrayOfStrings
Sets the column names to anArrayOfStrings. This can be used when the access type is named columns. The column names can also be set through the Properties dialog.

Returns: —
Triggers: —
**setColumnWidths: anArrayOfIntegers**
Sets the column widths to widths specified in `anArrayOfIntegers`. To initialize an array holder to an array of integers in the Workbench, you must enter a PARTStalk expression in the array holder’s New value: property. The following PARTStalk expression entered in the array holder’s New value: property field initializes the array holder to an array containing the integers 1, 2, 3, and 4:

#(1 2 3 4)

*Returns:* —
*Triggers:* —

**setFocus**
Moves the input focus to the table pane.

*Returns:* —
*Triggers:* —

**setFocusAtRow: aString or anInteger column: aString or anInteger**
Sets the focus to the specified row and column.

*Returns:* —
*Triggers:* —

**setNumberOfColumns: anInteger**
Sets the number of columns to `anInteger`.

*Returns:* —
*Triggers:* —

**setNumberOfRows: anInteger**
Sets the number of rows to `anInteger`.

*Returns:* —
*Triggers:* —

**setRowLabels: anArrayOfStrings**
Sets the row labels to `anArrayOfStrings`. The row labels can also be set through the Properties dialog. Labels support “named” access for indexable collections. A row or column with indexed access selected will generate label messages while a row or column with named access will generate name messages. The table pane will ignore labels assigned to named rows or columns and names assigned to indexed rows or columns. An automatically generated label message is indistinguishable from an automatically generated name message.

*Returns:* —
*Triggers:* —

**setRowNames: anArrayOfStrings**
Sets the row names to `anArrayOfStrings`. This can be used when the access type is named rows. The row names can also be set through the Properties dialog.

*Returns:* —
*Triggers:* —
setValue: aMatrix

Sets the contents of all the cells to the argument. aMatrix is anArrayOfArraysOfValues, or anArrayOfNamableCollectionsOfValues, or aNamableCollectionOfArraysOfValues, or aNamableCollectionOfNamableCollectionsOfValues.

Returns: —
Triggers:    cellChanged:
              cellChangedAtRow:column:value:

value

Returns all the values in the table. A Matrix is anArrayOfArraysOfValues, or anArrayOfNamableCollectionsOfValues, or aNamableCollectionOfArraysOfValues, or aNamableCollectionOfNamableCollectionsOfValues.

Returns: a Matrix
Triggers: —

Direct-Edit

Direct-editing a table pane lets you enter text into the cells and change the row and column labels.

To direct-edit a cell or label:

1 Alt <click> on the table pane.
2 Click in the cell or label you want to change.
3 Enter the new text.
4 Click in other cells to continue entering text in cells, if desired.
5 Click outside the table pane to terminate direct-editing.

When you direct-edit a column label, two new messages are automatically generated to access and set the contents of cells in that column, if you are using indexed access.
Property Edit

The Properties dialog for the table pane part is shown below. Each of the part-specific controls in the dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

The default values for the following four measurements are computed from the system font height and the display width.

**Cell width**
Sets the width of each column in the table.

**Cell height**
Sets the height of each row in the table.

**Number of rows**
Sets the number of rows initially in the table. The number of rows automatically increases when required by a message.

**Row label width**
Sets the width of each row label.

**Column label height**
Sets the height of each column label.

**Number of columns**
Sets the number of columns initially in the table. The number of columns automatically increases when required by a message.
Table Pane

**Access by**
Choose between Indexed or Named access for columns and rows.

**Adjust column widths**
If checked, the user can adjust the column width by dragging the dividing line. The default is to be able to adjust column widths.

**Adjust row heights**
If checked, the user can adjust the row height by dragging the dividing line. The default is to be able to adjust row heights.

**Display column labels**
If checked, the user can see column labels. When column labels are displayed the user can select a column by clicking on the label.

**Display row labels**
If checked, the user can see row labels. When row labels are displayed the user can select a row by clicking on the label.

**Read only**
If checked, the contents of the table cells can’t be changed by the user. The default is to allow the cell contents to be changed.

**Edit column labels**
If checked, the column labels can be changed at runtime. The default is to not be able to change the column labels at runtime.

**Edit row labels**
If checked, the row labels can be changed at runtime. The default is to not be able to change the row labels at runtime.

**Vertical scroll bar**
If checked, the table pane will have a vertical scroll bar.

**Horizontal scroll bar**
If checked, the table pane will have a horizontal scroll bar.
**Edit contents...**

This button opens a dialog (shown in the following illustration) which allows you to specify:
- Whether the column or row is read-only
- Column and row labels
- Column and row names
- Column widths based on character widths, and row heights in twips (where one twip is equal to 1/1440 of an inch)
- Forecolor, backcolor, and font of a row or column
- Format of a row or column including right justified text, left justified text, and right justified number

Note that row attributes have priority over column attributes.

The Default colors and font button for a row or column restores the overall default values specified for the entire table.

**Related Parts**
- **Entry Field**
  A table cell uses the same editing operations as an entry field.
Remarks

You can substitute other kinds of collections in place of Dictionaries as long they respond to the `at`, `atIfAbsent`, `atPut`, and `keysDo` Dictionary messages.

Programmatically Extending Table Panes

Table panes can be configured programatically in Smalltalk to achieve things that are currently not possible with the workbench. Typically, this involves configuring a table cell editor and having a table pane use this editor to edit the cells in a given row or column or use it for the entire table pane. Three methods in the class `PARTSTablePanePart` are used for this:

- `setColumn:`
  - `aColumn` editor: `aCellEditor`
  - Set `aCellEditor` to be the cell editor for the column at `aColumn`.

- `setRow:`
  - `aRow` editor: `aCellEditor`
  - Set `aCellEditor` to be the cell editor for the row at `aRow`.

- `setTableCellEditor:`
  - `aCellEditor` to be the cell editor for the entire table. Editors set for individual cells, columns, or rows will take precedence over this one.

Configuring a Table Cell Editor

Table cell editors are objects that display and edit the contents of a cell in a table pane. There are two approaches to configuring a table cell editor. The simple approach is to instantiate an existing table cell editor class and configure the result by sending messages to it. A less simple, but more powerful approach would be to subclass an existing table cell editor class, instantiate the subclass, and configure the result by sending messages to it.

Using an Existing Cell Editor Class

Existing cell editor classes support the following:

- Read only editors
- Aligning the text horizontally (left, center, or right)
- Aligning the text vertically (top, center, or bottom)
- Validating the inputted text
- Setting the foreground and background colors of a cell
- Setting the font with which the text will be displayed

If doing one or more of the above is all that you desire, then you do not have to create any new classes in this respect. There are two concrete classes used to instantiate the cell editors other than label editors:

- `PARTSTableCellTextEditor`
- `PARTSTableCellExtendedTextEditor`

The only attributes supported by a `PARTSTableCellExtendedTextEditor` that are not supported by a `PARTSTableCellTextEditor` are forecolor, backcolor, and font. A `PARTSTableCellTextEditor` uses the parent table pane’s forecolor, backcolor, and font.
There are three categories of messages you can send to a PARTSTableCellEditor to configure it: read only, text alignment, and text validation.

**Read only messages**

**readOnly: aBoolean**
Set the receiver to be read only, if aBoolean is true.

**Text alignment messages**

**setTextAlignBottom**
Configure the receiver to display and edit text at the bottom of the receiver's bounding box.

**setTextAlignCenteredHorizontally**
Configure the receiver to display and edit text centered horizontally.

**setTextAlignCenteredVertically**
Configure the receiver to display and edit text centered vertically.

**setTextAlignLeft**
Configure the receiver to display and edit text left justified.

**setTextAlignRight**
Configure the receiver to display and edit text right justified.

**setTextAlignTop**
Configure the receiver to display and edit text at the top of the receiver's bounding box.

**Validation messages**

**validationMessage: aMessage**
Set the message that will be evaluated to test if an input is acceptable. Prior to evaluation, this message's arguments will be set to an Array of one which contains the candidate new contents.

**Extended messages**

In addition to the read only, text alignment, and text validation messages that you can send to a PARTSTableCellTextEditor, with a PARTSTableCellExtendedTextEditor you can also support setting the foreground color, background color, and font. This is done with the following messages:

**backColor: aColor**
Set the receiver's backColor.

**font: aFont**
Set the receiver's font.

**foreColor: aColor**
Set the receiver's foreColor.
Sample scripts

Here is a script which can be attached to a table pane that illustrates how to modify column 2 of the table pane. This script should be invoked when the application is opened. It will cause the cells in column 2 of the table pane to be displayed and edited with their text justified to the top right corner, and the #IsValidInput: message will be sent to the table pane everyday a character is typed in a cell in column 2:

```
InitColumn2
"Configure an editor and associate it with column 2."
| e |
e := PARTSTableCellTextEditor new.
e
setTextAlignTop;
setTextAlignRight;
validationMessage: (Message
receiver: self
selector: #IsValidInput:
arguments: #()).
self setColumn: 2 editor: e
```

Here is the script that will get executed each time the user types in a cell in column 2 of a table pane that has executed the above script (InitColumn2). This will cause the cells in column 2 to only accept uppercase alphabetic characters:

```
IsValidInput: aString
"Answer whether aString is valid input."

aString do: [:c | c isUpperCase ifFalse: [
false]].

^true
```

The code used in these scripts can be used in Smalltalk methods as well.

Creating Subclasses of Cell Editor Classes

Subclasses of cell editor classes can be created to support a variety of things, such as cell editors that do not edit text, cell text editors with different display characteristics, cell text editors with more advanced text validation, cell text editors whose backing objects are not strings, and so on. It is beyond the scope of this document to be thorough on this subject, but here are some hints and pointers.

Different Cell Editor Display Characteristics

There is a subclass of PARTSTableCellExtendedTextEditor, PARTSTableLabelEditor, whose subclass's instances are the row, column, and corner labels of a table pane. These labels are displayed like a slightly raised button.

This raised effect is achieved by invoking the #display method in PARTSTableCellTextEditor (with 'super display') and then executing code to cause the raised effect (see PARTSTableLabelEditor>>display).

More Advanced Text Validation

The existing cell editor classes use a validation message to check if a user's keystroke should be accepted. If no validation message is set, the message #isLegalContentsDefault: will be sent to the cell editor. A subclass of an existing cell editor class could implement #isLegalContentsDefault: to perform a validation which for any reason should not be encapsulated in a message object.
Implementing `#isLegalContentsDefault` is still a limited form of validation in that it simply answers yes or no, accept or don’t accept the string presented. It cannot say “this is the string presented, but I will accept it as this string” (for example, converting the inputted string to upper case, or assuming the inputted string is an abbreviation and expanding it). To do something like this, you would implement the method `#insertString:` in your subclass.

**Backing Object Which Are Not Strings**

A cell editor can be designed to edit objects which are not strings. These objects are displayed and edited as strings, and the cell editor class has two methods that are key in managing this:

`#convertToString: anObject`

Answers a string representation of the backing object, which is the argument to the method.

`#convertFromString: aString`

Answers the backing object that is encoded in the argument string.

A subclass of a cell editor class can implement just these two methods to enable editing of any type of object.

**Example**

This example lets you try out the selection and editing mechanisms of a table pane. The `setOne:` entry field lets you supply new values for cells in column one. Use the RowUp, RowDown, RowTop, and RowBottom buttons to change the row of the cell set by `setOne:`.
Table Pane

The values of many of the table pane events are displayed below the table pane. Try clicking, double-clicking, and dragging to select individual cells and groups of cells.

This example is in the file TBLPXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
Text Pane

The text pane is a visual part that lets you enter and edit multiple lines of text. The contents of a text pane can be read from and stored to a file or changed at runtime with messages or user actions. It has a modified flag that is set whenever the text changes. This flag can be tested and cleared.

The following PARTS Workbench Interface window has a text pane at the bottom for editing scripts.

Operation

Text is entered the same as in an entry field, except that pressing the Enter key just starts a new line of text.

Entering or modifying text sets the modified flag. The text pane can be tested to see if its contents have been modified by sending the `isModified` message. If the text pane has been modified, the `modifiedIsTrue` event is triggered. After the contents have been saved, the `clearModified` message can be used to reset the modified flag.

There are two different ways to save changes to a text pane:

- Detect whether the contents have changed by sending the `isModified` message when the pane is closed. If the `modifiedIsTrue` event triggers, the contents have been modified. The `modifiedIsTrue` event can be linked to either the `value` or `saveToFile:` messages to preserve the changes.

- Link the `changed:` event to a message. The `changed:` event triggers whenever the input focus leaves the text pane and changes have been made. The event value provided is a string containing the new contents of the text pane. The modified flag is cleared whenever the `changed:` event triggers the execution of
a message. If the changed: event is not linked to a message, the modified flag
is not cleared.

If these two techniques for recording changes are used together, the
modifiedIsTrue event will only be triggered if the isModified message is executed
after changes have been made to the text pane, but before the input focus leaves it.

Events

changed: aString
Occurs when the text pane loses input focus and the contents have been
changed since the input focus entered the text pane, or when a message is
executed that replaces the contents of the text pane. The event value is the
current contents. The modified flag is cleared whenever this event triggers
the execution of a message.

Triggered By: the user
clear
deleteKey
setFromFile:
setValue:

controlTabbed
Occurs when the Control and Tab keys are pressed simultaneously and the
text pane has the input focus. You can connect this event to the setFocus
message of another part to set the tab order.

Triggered By: the user

fileException: aString
Occurs when a file cannot be accessed. The event value contains the
pathname of the file.

Triggered By: saveToFile:
setFromFile:

modifiedIsTrue
Occurs when the text pane is sent the isModified message and the modified
flag is set.

Triggered By: isModified

rightClicked
Occurs when the user clicks mouse button 2 on the text pane.

Triggered By: the user

Messages

append: aString
Appends the argument to the text in the pane and sets the modified flag.

Returns: —
Triggers: —

clear
Deletes all of the text in the text pane and clears the modified flag.

Returns: —
Triggers: changed:
clearModified
Clears the modified flag.

Returns: —
Triggers: —

clearSelection
Deletes the selected text from the text pane and sets the modified flag.

Returns: —
Triggers: —

copySelection
Copies the currently selected text to the system clipboard.

Returns: —
Triggers: —
cutSelection
Copies the currently selected text to the system clipboard and then deletes it from the text pane. The modified flag is set. This is the same as copySelection followed by clearSelection.

Returns: —
Triggers: —
deleteKey
Simulates pressing the Delete key.

Returns: —
Triggers: changed:
disable
Ignores user input.

Returns: —
Triggers: —
enable
Enables the text pane to respond to user input.

Returns: —
Triggers: —
insert: aString
Replaces the current selection with the argument. The argument is inserted at the current input point if there is no text selection. The modified flag is set.

Returns: —
Triggers: —
isModified
Triggers modifiedIsTrue if the modified flag is set and returns true. Returns false if the modified flag is not set. The modified flag is not changed.

Returns: a Boolean
Triggers: modifiedIsTrue
pasteSelection
Replaces the current selection with the text in the system clipboard. The clipboard text is inserted at the current input point if there is no text selection. The modified flag is set.

Returns: —
Triggers: —

saveToFile: aString
Saves the contents of the text pane to the file whose path name is given by the argument. The modified flag is not affected.

Returns: —
Triggers: fileException:

selectAll
Selects all the text in the text pane.

Returns: —
Triggers: —

selectedText
Returns the currently selected text. An empty string is returned if no text is selected.

Returns: a String
Triggers: —

setFocus
Moves the input focus to the text pane.

Returns: —
Triggers: —

setFromFile: aString
Sets the contents of the text pane from the contents of the file whose path name is given by the argument. The modified flag is cleared.

Returns: —
Triggers: changed
fileException:

setModified
Sets the modified flag.

Returns: —
Triggers: —

setValue: aString
Sets the contents of the pane to the argument and clears the modified flag.

Returns: aString
Triggers: changed:

undo
Undo the last user edit in the text pane. Sets the modified flag and triggers the changed: event if the undo operation changes the text.

Returns: —
Triggers: —
value
Returns the contents of the text pane.

Returns: a String
Triggers: —

Direct-Edit
Direct-editing a text pane lets you enter the initial contents of the text pane.

Property Edit
Each of the part-specific controls in the text pane’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Styles
The following options can be selected:

- **Word wrap**
  If checked, the text displayed is wrapped so it is all visible in the pane. Otherwise the text must be scrolled horizontally if the line is wider than the pane. The default is to have word wrap on.

- **Read only**
  If checked, the contents of the text pane can’t be edited at runtime. The default is to allow editing of the contents.

- **Border**
  If checked, a border appears around the text pane. The default is to display the border.

- **Vertical scroll bar**
  If checked, the vertical scroll bar is displayed. The default is to display the vertical scroll bar.

- **Horizontal scroll bar**
  If checked, the horizontal scroll bar is displayed. The default is to not display the horizontal scroll bar.
**Edit contents...**

This button opens the dialog shown below. Enter the initial contents of the text pane and press OK.

![Edit contents dialog](image)

**Drag drop...**

Opens the following dialog where you can enable the capability for objects to be dragged from and/or dropped on the part:

![Drag drop dialog](image)

The Source and the Target check boxes in the Drag group pane are used to enable (or disable) the part as the source and/or the target for a specified set of drag and drop operations.

The Unused and the Chosen and Prioritized list panes in the Drag Target Formats group pane are used to indicate how the part should render a dropped object, and they are disabled unless the part has been enabled as a target. This target part attempts to render the dropped object in one of the chosen formats, beginning with the first entry in the Chosen and Prioritized list pane. If the object cannot be rendered in that format, then the other chosen formats are tried. If the object cannot be rendered in any of the chosen formats, then it is ignored. To move formats
between the Unused and the Chosen and Prioritized list panes, drag them from one list to the other. To change the priority order of the chosen formats, drag them within the Chosen and Prioritized list pane.

The Select Operations multiple-selection list pane in the Drag Target Operations group pane is used to indicate the set of drag and drop operations that the part will accept, and it is disabled unless the part has been enabled as a target. An operation is selected if it is highlighted.

The Defaults button is used to restore the part’s drag drop property to its default values.

**Related Parts**

- **Edit Menu**
  You can hook the edit menu `popup` message to the `rightClicked` event or add an Edit menu label, and then connect the menu items in the edit menu to the text pane messages to add an edit menu to any text pane.

- **Entry Field**
  Lets you enter a single line of text.

**Example**

This example lets you try out some of the events and message of a text pane. After entering text into the text pane you can press the value button to retrieve the contents of the text pane and display them in the adjacent entry field. Or, enter text into the entry field and press Return and `setValue` will be used to set the text pane contents to the entry field contents.

Pressing the isModified? button sends the `isModified` message, the result of which is displayed in the adjacent static text part.
The **changed:** event is not linked in this example so you can see the default behavior of the text pane. When this event is not used, the modified flag is cleared only when the **clearModified**, **clear**, or **setValue:** messages are executed.

The open... and save... buttons bring up file dialogs so you can set the text pane contents from a file, or save them to a file.

This example is in the file TXTPXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The timer is a nonvisual part that triggers events whenever the timer interval expires. The timer triggers the **ticked** and **tickedTime:** events. The timer can be started and stopped with the messages `startTimer` and `stopTimer`. The timer interval is initially 1000 milliseconds (one second), but can be set to any number of milliseconds in the Properties dialog or with the `setInterval:` message.

Once started, the timer keeps running until the `stopTimer` message is executed.

**Events**

**ticked**  
Occurs whenever the timer is running and the timer interval has expired.  
*Triggered By:* `startTimer`

**tickedTime:** `aTime`  
Occurs whenever the timer is running and the timer interval has expired. The current time is provided by the event value.  
*Triggered By:* `startTimer`

**Messages**

**currentDate**  
Returns the current date.  
*Returns:* a Date  
*Triggers:* —

**currentTime**  
Returns the current time.  
*Returns:* a Time  
*Triggers:* —

**setInterval:** `anInteger`  
Changes the timer interval to `anInteger` milliseconds. If the timer was running, it is stopped and restarted with the new interval.  
*Returns:* `anInteger`  
*Triggers:* `ticked`  
`tickedTime:`

**startTimer**  
Starts the timer running and triggering events whenever the timer interval elapses.  
*Returns:* —  
*Triggers:* `ticked`  
`tickedTime:`
Timer

**stopTimer**

Stops the timer from triggering events.

*Returns:* —

*Triggers:* —

**Direct-Edit**

Direct-editing the timer changes the part name that appears beneath the icon in the workbench.

**Property Edit**

The part-specific control in the timer's Properties dialog is described in the following paragraph. For an explanation of any of the standard properties, refer to *Appendix C, Standard Part Properties*.

![Timer Properties Dialog](image)

**Interval in milliseconds**

This field lets you change the interval (in milliseconds) between ticks of the timer. Enter the desired timer interval and press OK. The default interval is 1000 milliseconds, which is equivalent to one second.

**Remarks**

The **tickedTime:** event is much less efficient than **ticked** because it accesses the system clock to obtain the current time every time the event triggers. You should only use **tickedTime:** if you need the current time as the event value.

Your application should stop all timers before it terminates. Otherwise they may continue to use system resources.
Example

This example is a digital clock. Press the Start button to start the clock running. Every time the `tickedTime` event triggers, the current time is displayed in the static text part. Pressing the Stop button stops the clock.

This example is in the file TIMRXMPL.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
**Tool Box**

*(Win32 only)*

A tool box is a part that is used to hold tools for a toolbar. It has sections for the tools that are initially displayed in the toolbar and for any additional tools that the user can add to the toolbar.

A subpane (such as a Combo Box or a Drop-down List) can appear on a toolbar if it exists in the tool box.

**Operation**

Drop a toolbar onto a window. By default, it will appear just below the menu bar of the window. Place a tool box on the background of the workbench. Set the contents of the toolbar from the toolbox, using either of the following methods:

- Link the toolbar’s `needsContents` event to its `setItems:` message, where the argument is the `tools` message of the tool box.
- Link the toolbar’s `needsContents` event directly to the `tools` message of the tool box.

Tools and subpanes are dropped into the tool box (rather than directly on the toolbar). Note that tools will be shown in the toolbar in the order that they are arranged in their corresponding tool box. A tool or subpane will not be initially displayed in the toolbar if it is placed in the Detached from Toolbar section of the tool box.

The items in the tool box will wrap to fit in the tool box. You can also manually resize the tool box.

**Events**

There are no events for a tool box.

**Messages**

A tool box has the following message.

```smalltalk
tools

Returns: a Collection of tools
Triggers: —
```


Tool Box (Win32 only)

Direct-Edit

There is no direct-editing for the tool box. If you attempt to direct-edit this part, you receive the following dialog:

Property Edit

There are no part-specific properties for a tool box. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.

Related Parts

- **ToolBar (Win32 only)**
  The toolbar is a visual part that is conceptually similar to a menu, and has buttons providing quick access to frequently-used commands.

- **ToolBar Button (Win32 only)**
  The toolbar button is a visual part with a bitmap image and/or a text label indicating its function, that the user clicks on to initiate an action.

- **ToolBar Separator (Win32 only)**
  The toolbar separator is a visual part that is used in a toolbar to separate groups of toolbar buttons and toolbar toggle buttons.

- **ToolBar Toggle Button (Win32 only)**
  The toolbar toggle button is a visual part similar to a toolbar button, but with different actions for its on and off states.

Remarks

A toolbar is conceptually similar to a menu. Many graphical applications provide a toolbar of frequently-used commands, which are also available from a menu.

Example

See the example in the Toolbar (Win32 only) chapter later in this document.
**Toolbar**

*(Win32 only)*

A toolbar is a visual part that is conceptually similar to a menu, and has buttons (with bitmap images and/or text labels) providing quick access to frequently-used commands. The user can change the toolbar if its Customizable property is enabled.

A toolbar contains a combination of the following components:

- Toolbar button (similar to a drawn button)
- Toolbar toggle button (similar to a drawn button, but also having on and off states with different actions and appearances)
- Toolbar separator (similar to a menu item separator)

A subpane (such as a Combo Box or a Drop-down List) can appear on a toolbar if it exists in the tool box. A tool box is a related part that is used to hold tools for the toolbar.

The following illustration of the Windows 95 Explorer demonstrates the use of a toolbar:

---

**Operation**

**Adding a Toolbar to a Window**

Drop a toolbar onto a window. By default, it will appear just below the menu bar of the window. Place a tool box on the background of the workbench. Set the contents of the toolbar from the toolbox, using either of the following methods:

- Link the toolbar’s `needsContents` event to its `setItems:` message, where the argument is the `tools` message of the tool box.
- Link the toolbar’s `needsContents` event directly to the `tools` message of the tool box.
Tools and subpanes are dropped into the tool box (rather than directly on the toolbar). Note that tools will be shown in the toolbar in the order that they are arranged in their corresponding tool box. A tool or subpane will not be initially displayed in the toolbar if it is placed in the Detached from Toolbar section of the tool box.

**Customizing a Toolbar**

If the Customizable property is set for a toolbar, you can customize it in the running application either by holding down the Shift key (or the Alt key, if the Drag with Alt key property is also set) while dragging its tools to the desired arrangement, or by double-clicking on the background of the toolbar and then using the following dialog:

![Customize Toolbar Dialog](image)

The Add button adds the selected available button or separator to the toolbar, while the Remove button detaches the selected button or separator from the toolbar and moves it to the available list. Changes are applied immediately to the toolbar in the running application, so you can see the results while this dialog is still open.

**Saving and Restoring the State of a Toolbar's Tools**

The state of the toolbar’s tools can be saved or restored using the Registration Database. To use this functionality, you must specify a value for the Registration Key Name property. The key name is composed of a string containing backward slashes (for example, 'SOFTWARE\CompanyName\ApplicationName').

You can also specify a value for the Registration Value Name property. The value name is a string (such as 'Favorite Settings' or 'App1 Config') that can be used to identify different configurations of the toolbar in a particular application. The registration entry will be saved under the heading HKEY_LOCAL_MACHINE.

If the registration database entry does not exist, it will be created.

When the toolbar opens and the registration database information is set, it will restore its tool state from that entry. If the Auto Save On Close property is checked, the state of the toolbar is automatically saved to the registration database entry when the toolbar is closed. The **save** and **restore** messages can be used at any time to save or restore the toolbar’s state.

If your application uses this functionality of the Registration database, you must include the VREG311W.SLL support library when delivering the application.
Events

A toolbar has the following events.

**beginAdjust**
Triggered when the toolbar is about to be customized or when its contents are about to be restored.

*Triggered By:* the user
- customize
- restore

**changed**
Triggered when the toolbar changes, such as when the user customizes it.

*Triggered By:* the user

**clicked:** a tool
Triggered when the user clicks on one of the tools in the toolbar. The argument is the tool that was clicked.

*Triggered By:* the user

**doubleClicked**
Triggered when the user double-clicks on the toolbar itself (not one of its tools).

*Triggered By:* the user

**endAdjust**
Triggered when the toolbar has been customized or when its contents have been stored.

*Triggered By:* the user
- save

**needsContents**
Occurs when the toolbar is about to be opened, to request that its contents be set.

**reset**
Triggered when the contents of the toolbar are reset by the user clicking on the Reset button in the Customize Toolbar dialog.

*Triggered By:* the user
A toolbar has the following messages.

autoSaveOnClose: aBoolean

Specifies whether the state of the toolbar is to be automatically saved (in the Registration database) when the toolbar is closed. (This can also be specified with the Auto Save On Close property.)

Returns: aBoolean
Triggers: —

customize

Customize the receiver by displaying the Customize Toolbar dialog box. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: beginAdjust
disable

Disables the toolbar so it does not accept user input.

Returns: —
Triggers: —

enable

Enables the toolbar so it accepts user input.

Returns: —
Triggers: —

hide

Prevents the display of the toolbar.

Returns: —
Triggers: —

items

Returns all the tools in the toolbar.

Returns: a Collection of tools
Triggers: —

registrationKeyName

Returns the key name of the Registration database entry where the state of the toolbar is saved.

Returns: a String
Triggers: —

registrationKeyName: aString

Sets the key name of the Registration database entry where the state of the toolbar will be saved.

Returns: —
Triggers: —
registrationValueName
   Returns the value name of the Registration database entry where the state of the toolbar is saved.
   
   Returns: a String
   Triggers: —

registrationValueName: aString
   Sets the value name of the Registration database entry where the state of the toolbar will be saved.
   
   Returns: —
   Triggers: —

setItems: aToolCollection
   Sets the contents of the toolbar from a tool box.
   
   Returns: —
   Triggers: —

show
   Displays the toolbar.
   
   Returns: —
   Triggers: —

Direct-Edit

There is no direct-editing for the toolbar. If you attempt to direct-edit this part, you receive the following dialog:

![Direct-Edit dialog](image)

Property Edit

Each of the part-specific controls in the toolbar's Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.

![Property Edit dialog](image)
**Customizable**
This check box (which is checked by default) indicates whether the user can customize the toolbar. When this box is checked, the user can change the position of a tool in the toolbar by holding down the Shift key (or the Alt key, if the Drag with Alt key property is also set) while dragging it to the desired location, and can double-click on the toolbar to pop up a customization dialog.

**Drag with Alt key**
This check box indicates whether the user can use the Alt key (instead of the Shift key) when dragging a tool on a customizable toolbar to the desired location.

**Place At Top**
This check box (which is checked by default) indicates whether the toolbar is placed at the top of the window below the menu bar. If this box is not checked, the toolbar is placed at the bottom of the window.

**Tool Tip**
This check box indicates whether the display of tip text is enabled for the tools on the toolbar. When this capability is enabled, tip text for a specific tool is displayed after the user leaves the cursor positioned over the tool.

**Wrapable**
This check box indicates whether the toolbar can wrap onto additional lines (instead of being truncated) if it is unable to fit within the current width of the window. Note that setting this property overrides the Wrapped property of buttons on the toolbar.
**Bitmap Height**
This integer entry field specifies the height (in pixels) of the optional tool bitmaps appearing on the toolbar. The default value is 15 pixels.

**Bitmap Width**
This integer entry field specifies the width (in pixels) of the optional tool bitmaps appearing on the toolbar. The default value is 16 pixels.

Bitmaps larger than the standard size set here are cropped.

**Minimum Tool Height**
This integer entry field specifies the minimum height (in pixels) of the tools on the toolbar. The default value is 22 pixels.

**Minimum Tool Width**
This integer entry field specifies the minimum width (in pixels) of the tools on the toolbar. The default value is 24 pixels.

Note that the size of the tools is automatically increased to hold the longest text label or combination of bitmap and text label.
Registration Key Name
This text entry field is used to specify a value for the registration key name, which identifies a group of entries in the registration database that hold the state of the toolbar. The key name is a string containing backward slashes (such as ‘SOFTWARE\CompanyName\ApplicationName’). If the registration database entry does not exist, it will be created.

Registration Value Name
This text entry field is used to specify a value for the registration value name, which identifies the entry in the registration database that holds a particular state of the toolbar. The value name is a string (such as ‘Favorite Settings’ or ‘App1 Config’) that can be used to identify different configurations of the toolbar in a particular application. The registration entry will be saved under the heading HKEY_LOCAL_MACHINE. When the toolbar opens and the registration database information is set, it will restore its tool state from that entry.

Auto Save On Close
This check box determines whether the state of the toolbar is automatically saved when the toolbar is closed.

Related Parts
- **Tool Box (Win32 only)**
  The tool box is a part that is used to hold tools for a toolbar.
- **Toolbar Button (Win32 only)**
  The toolbar button is a visual part with a bitmap image and/or a text label indicating its function, that the user clicks on to initiate an action.
- **Toolbar Separator (Win32 only)**
  The toolbar separator is a visual part that is used in a toolbar to separate groups of toolbar buttons and toolbar toggle buttons.
- **Toolbar Toggle Button (Win32 only)**
  The toolbar toggle button is a visual part similar to a toolbar button, but with different actions for its on and off states.

Remarks
A toolbar is conceptually similar to a menu. Many graphical applications provide a toolbar of frequently-used commands, which are also available from a menu.
Example

The previous illustration shows a modified version of the EDITOR.PAR sample application, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. This sample text-editing application demonstrates the combined use of many parts, including:

- Toolbar (Win32 only), Toolbox (Win32 only), Toolbar Button (Win32 only), and Toolbar Separator (Win32 only)
- Rich Edit (Win32 only), Format Menu (Win32 only), File Menu, and Edit Menu
- Status Window (Win32 only) and Timer

For additional explanation of selected component parts in this sample application, refer to the example in the following chapters elsewhere in this document:

- Format Menu (Win32 only)
- Rich Edit (Win32 only)
- Status Window (Win32 only)
- Toolbar Button (Win32 only)
Toolbar (Win32 only)
A toolbar button is a visual part with a bitmap image and/or a text label indicating its function, that is operated by clicking on it. A toolbar button is one of the components of a toolbar, which is conceptually similar to a menu but has buttons providing quick access to frequently-used commands.

Pressing a toolbar button usually initiates an action such as formatting selected text, saving a file, or printing. Inactive buttons can be dimmed.

The following illustration of the Windows 95 Explorer demonstrates the flexibility of toolbar buttons and toolbar toggle buttons appearing on a toolbar:

**Operation**

When building a toolbar, drop its buttons and separators into a tool box rather than onto the toolbar itself.

To create a group of toolbar buttons, place them next to each other in the desired order within the tool box, and set the Grouped property for each of the buttons in the group. If your toolbar has more than one group, toolbar separators are required to differentiate the groups. (Although a toolbar separator is not required if your toolbar has only one group of buttons, it is good practice to visually separate the group from other items appearing on the toolbar.)

To operate a toolbar button in the running application, click on it. The border and background color of the toolbar button change to indicate that the button is pressed. The **clicked** event is triggered when you release the mouse button.
ToolBar Button (Win32 only)

Events

clicked
Occurs when the user clicks on the button, or the click message is sent.

Triggered By: the user
                click

Messages

bitmap
Returns the optional bitmap image appearing on the button.

Returns: a Bitmap
Triggers: —

clearIndeterminate
Clears the indeterminate state of the button.

Returns: —
Triggers: —

click
Programmatically clicks the button.

Returns: —
Triggers: clicked

detach
Removes the button from the view of the user and stores it in the Available buttons list of the Customize Toolbar dialog, where it can be retrieved later. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

disable
Causes the button to ignore user input and dims the button. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

enable
Enables the button to respond to user input and restores the original appearance of the button. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

hide
Removes the button from the view of the user and stores it (with a dithered appearance) in the Toolbar buttons list of the Customize Toolbar dialog. The show message can be used to make the button reappear in the toolbar. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —
label

Returns the button's optional text label.

Returns: a String
Triggers: —

makelndeterminate

Sets the button's state to indeterminate, where it is visible and accepts user input but does not produce any events or actions.

Returns: —
Triggers: —

show

Shows the button in the toolbar, after it had been hidden by the hide message. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

tip

Answers the optional tool tip text for the button.

Returns: a String
Triggers: —

Direct-Edit

Direct-editing a toolbar button opens the standard bitmap editor.

Property Edit

Each of the part-specific controls in the toolbar button's Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.
Grouped

This check box indicates whether the button is part of a group, where only one button in the group can be selected at a time. (When clicked on by the user, the button stays pressed until another button in the group is clicked on.)

If your toolbar has more than one group, toolbar separators are required to differentiate the groups.

Wrapped

This check box indicates whether the toolbar is wrapped onto another line immediately following this button. Setting the Wrapable property of the toolbar containing this button overrides this Wrapped property of the button.

Enabled

This check box (which is checked by default) indicates whether the button is initially enabled to accept user input. You can uncheck this box to have the button initially disabled. For example, a button used to apply formatting to selected text could be disabled until some text has been selected.
Hidden

This check box indicates whether a tool is initially hidden from the user's view. A hidden tool is similar to a detached tool, except that a hidden tool is initially placed in the In Toolbar section of the tool box and is shown (with a dithered appearance) in the Toolbar buttons list of the Customize Toolbar dialog. The tool can be made visible with the show message.

A tool can initially be detached by being placed in the Detached from Toolbar section of the tool box. A tool that was not initially detached can become detached with the detach message. In a customizable toolbar, a tool can also be detached by being dragged past the edge of the toolbar or by double-clicking the toolbar and using the Customize Toolbar dialog.

Indeterminate

This check box indicates whether the button is in the indeterminate state, where it is visible and accepts user input but does not produce any events or actions.

Label

This text entry field allows you to specify an optional text label that will appear on the button. (If an optional bitmap image is supplied, then the text label appears below the image.)

Tip

This text entry field is where you supply the tip text for this tool. This tip text is used only if the Tool Tip property of the toolbar is checked.

Image

This button opens the standard bitmap editor, so you can create or specify an optional bitmap image that will appear on the button. (If an optional text label is supplied, it appears below the image.) An image that is larger than the size set with the Bitmap Height and Bitmap Width properties of the toolbar will be cropped.
The following illustration shows the interaction of the Label and Image properties:

![Illustration of Label and Image properties](image)

**Related Parts**

- **Push Button**
  The push button is a visual part (with a label describing its function) that the user clicks on to initiate an action.

- **Tool Box (Win32 only)**
  The tool box is a part that is used to hold tools for a toolbar.

- **Toolbar (Win32 only)**
  The toolbar is a visual part that is conceptually similar to a menu, and has buttons providing quick access to frequently-used commands.

- **Toolbar Separator (Win32 only)**
  The toolbar separator is a visual part that is used in a toolbar to separate groups of toolbar buttons and toolbar toggle buttons.

- **Toolbar Toggle Button (Win32 only)**
  The toolbar toggle button is a visual part similar to a toolbar button, but with different actions for its on and off states.

**Example**

The toolbar button is used in the EDITOR.PAR sample application, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. The toolbar buttons provide the same functionality as certain menu commands, and their tip text is displayed in the status window at the bottom of the application.

For further explanation of this sample application, see the example in the Toolbar (Win32 only) chapter earlier in this document.
Toolbar Separator
(Win32 only)

The toolbar separator is a visual part that you drop into a tool box to separate groups of toolbar buttons and toolbar toggle buttons. (It is equivalent to the menu separator used to separate groups of items in a menu.)

The toolbar separator appears in the toolbar as a region of blank space. The following illustration of the Windows 95 Explorer demonstrates the use of toolbar separators to group related toolbar buttons and toolbar toggle buttons:

In general, the toolbar separator is an optional item that can improve the appearance and usability of a toolbar. However, a toolbar separator is required to create more than one set of grouped buttons (with the Grouped property checked) in your toolbar.

Events

A toolbar separator has no events.

Messages

A toolbar separator has no messages.

Direct-Edit

There is no direct-editing for the toolbar separator. If you attempt to direct-edit this part, you receive the following dialog:
Property Edit

The part-specific control in the toolbar separator’s Properties dialog is described in the following paragraph. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.

Width

This integer entry field specifies the width (in pixels) of this toolbar separator. The default value is 5 pixels.

Related Parts

- **Menu Separator**
  The menu separator is a line that separates two adjacent menu items when you drop it between them in a menu.

- **Tool Box (Win32 only)**
  The tool box is a part that is used to hold tools for a toolbar.

- **Toolbar (Win32 only)**
  The toolbar is a visual part that is conceptually similar to a menu, and has buttons providing quick access to frequently-used commands.

- **Toolbar Button (Win32 only)**
  The toolbar button is a visual part with a bitmap image and/or a text label indicating its function, that the user clicks on to initiate an action.

- **Toolbar Toggle Button (Win32 only)**
  The toolbar toggle button is a visual part similar to a toolbar button, but with different actions for its on and off states.
Example

The toolbar separator is used in the EDITOR.PAR sample application, located in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. For further explanation of this sample application, see the example in the Toolbar (Win32 only) chapter earlier in this document.
Toolbar Separator (Win32 only)
Toolbar Toggle Button
(Win32 only)

A toolbar toggle button is a visual part, with a bitmap image and/or a text label indicating its function, that is operated by clicking on it. The toolbar toggle button is a special type of toolbar button with different actions for its on and off states, and an appearance that changes according to its state.

Pressing a toolbar toggle button toggles its current value between on (when it is pressed) and off (when it is not pressed). A toolbar toggle button is commonly used for applying or removing a certain kind of formatting from selected text.

Toolbar toggle buttons can simulate either independent check boxes or grouped radio buttons. The following illustration of the Windows 95 Explorer demonstrates this second type of usage, with the toggle for Large Icons indicating the type of view shown in the right-hand pane:

![Illustration of Windows 95 Explorer with toolbar toggle buttons for Large Icons]

Operation

When building a toolbar, drop its buttons and separators into a tool box rather than onto the toolbar itself.

To create a group of toolbar toggle buttons, place them next to each other in the desired order within the tool box, and set the Grouped property for each of the buttons in the group. If your toolbar has more than one group, toolbar separators are required to differentiate the groups. (Although a toolbar separator is not required if your toolbar has only one group of buttons, it is good practice to visually separate the group from other items appearing on the toolbar.)

To operate a toolbar toggle button in the running application, click on it. The border and background color of the toolbar toggle button change to indicate whether the button is pressed. When you release the mouse button, the clicked: event and one of the turnedOff or turnedOn events are triggered. The clicked: event has a Boolean argument that indicates the state of the toolbar toggle button; true if pressed, false if not pressed.
Events

**clicked:** aBoolean

Occurs when the user clicks on the button, or the **click** message is sent.
aBoolean indicates the state of the toolbar toggle button; true if pressed, false
if not pressed.

Triggered By: the user
  click

**turnedOff**

Occurs when the user clicks on the button and its state is turned on, or the
**turnOff** message is sent.

Triggered By: the user
  turnOff

**turnedOn**

Occurs when the user clicks on the button and its state is turned off, or the
**turnOn** message is sent.

Triggered By: the user
  turnOn

Messages

**bitmap**

Returns the optional bitmap image appearing on the button.

Returns: a Bitmap
  Triggers: —

**clearIndeterminate**

Clears the indeterminate state of the button.

Returns: —
  Triggers: —

**click**

Programmatically clicks the button.

Returns: —
  Triggers: clicked:
  turnedOff
  turnedOn

**detach**

Removes the button from the toolbar. (The tool remains in the tool box,
however, so it can be added back to the toolbar.) The return value indicates
whether the operation was successful.

Returns: a Boolean
  Triggers: —

**disable**

Causes the button to ignore user input and dims the button. The return value
indicates whether the operation was successful.

Returns: a Boolean
  Triggers: —
enable
Enables the button to respond to user input and restores the original appearance of the button. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

hide
Prevents the display of the toolbar. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

isOn
Answers whether the button’s state is on.

Returns: a Boolean
Triggers: —

label
Returns the button’s optional text label.

Returns: a String
Triggers: —

makelneterminate
Sets the button’s state to indeterminate, where it is visible and accepts user input but does not produce any events or actions.

Returns: —
Triggers: —

show
Displays the toolbar. The return value indicates whether the operation was successful.

Returns: a Boolean
Triggers: —

tip
Answers the optional tool tip text for the button.

Returns: a String
Triggers: —

turnOff
Sets the button’s state to off.

Returns: —
Triggers: changed turnedOff

turnOn
Sets the button’s state to on.

Returns: —
Triggers: changed turnedOn
Direct-Edit

Direct-editing a toolbar toggle button opens the standard bitmap editor.

Property Edit

Each of the part-specific controls in the toolbar toggle button's Properties dialog is described in the following paragraphs. For an explanation of any of its standard properties, refer to Appendix C, Standard Part Properties.

Grouped

This check box indicates whether the button is part of a group, where only one button in the group can be selected at a time. (When clicked on by the user, the button stays pressed until another button in the group is clicked on.)

If your toolbar has more than one group, toolbar separators are required to differentiate the groups.

Wrapped

This check box indicates whether the toolbar is wrapped onto another line immediately following this button. Setting the Wrapable property of the toolbar containing this button overrides this Wrapped property of the button.
Enabled
This check box (which is checked by default) indicates whether the button is initially enabled to accept user input. You can uncheck this box to have the button initially disabled. For example, a button used to apply formatting to selected text could be disabled until some text has been selected.

Hidden
This check box indicates whether a tool is initially hidden from the user’s view. A hidden tool is similar to a detached tool, except that a hidden tool is initially placed in the In Toolbar section of the tool box and is shown (with a dithered appearance) in the tool box and in the Toolbar buttons list of the Customize Toolbar dialog. The tool can be made visible with the show message.

A tool can initially be detached by being placed in the Detached from Toolbar section of the tool box. A tool that was not initially detached can become detached with the detach message. In a customizable toolbar, a tool can also be detached by being dragged past the edge of the toolbar or by double-clicking the toolbar and using the Customize Toolbar dialog.

Indeterminate
This check box indicates whether the button is in the indeterminate state, where it is visible and accepts user input but does not produce any events or actions.

Turned On
This check box indicates whether the button is initially turned on, so that it appears to be pressed. For example, you might want to initially turn on one of the buttons in a group.
ToolBar Toggle Button (Win32 only)

Label
This text entry field allows you to enter an optional text label that will appear on the button. (If an optional image is supplied, then the text label appears below the image.)

Tip
This text entry field is where you supply the tip text for this tool. This tip text is used only if the Tool Tip property of the toolbar is checked.

Image
This button opens the standard bitmap editor, so you can create or specify an optional bitmap image that will appear on the button. (If an optional text label is supplied, it appears below the image.) An image that is larger than the size set with the Bitmap Height and Bitmap Width properties of the toolbar will be cropped.

The following illustration shows the interaction of the Label and Image properties:

Related Parts

- **ToolBar (Win32 only)**
  The tool box is a part that is used to hold tools for a toolbar.

- **Toolbar (Win32 only)**
  The toolbar is a visual part that is conceptually similar to a menu, and has buttons providing quick access to frequently-used commands.

- **ToolBar Button (Win32 only)**
  The toolbar button is a visual part with a bitmap image and/or a text label indicating its function, that the user clicks on to initiate an action.
• **Toolbar Separator (Win32 only)**
  The toolbar separator is a visual part that is used in a toolbar to separate groups of toolbar buttons and toolbar toggle buttons.

**Example**

See the example in the *Toolbar (Win32 only)* chapter earlier in this document.
Toolbar Toggle Button (Win32 only)
A track bar is a window containing a slider control, which the user can move to select a single value within a range of integer values. A track bar has tick marks indicating the increments within its range, and can be oriented either horizontally or vertically.

This track bar part is similar to the Sliders (OS/2 only) and Scroll Bar parts. You can use a track bar to allow a user to visually set values that have familiar increments, such as feet, inches, and degrees. It can also be used when immediate visual feedback is necessary, such as an application that allows users to blend colors.

Operation

You define a track bar's range of values and its initial value by setting its minimum, maximum, and initial values in its Properties dialog.

You can operate the track bar in the following ways:

- Send a message (such as `incrementPosition`, `nextPage`, or `setValue`) to the track bar
- Drag the slider bar
- For the Windows version of this part, click on the shaft of the track bar (on either side of the slider bar) to move by the page increment in the direction of the click

Events

A track bar has the following events.

**changed: anInteger**

The current position of the slider was changed, either by the user or programmatically.

_Trigged By:_ the user

**changing: anInteger**

The user is changing the value by dragging the slider.

_Trigged By:_ the user

**end**

The user pressed the End key to jump to the maximum value.

_Trigged By:_ the user

**home**

The user pressed the Home key to jump to the minimum value.

_Trigged By:_ the user
nextLine
The user pressed the right arrow key (for a horizontal track bar) or the up
arrow key (for a vertical track bar) to increment the current value by the
amount of the current line increment.

Triggered By: the user

nextPage
The user pressed the PageUp key, or clicked to the right of (for a horizontal
track bar) or above (for a vertical track bar) the slider bar, to increment the
current value by the amount of the current page increment.

Triggered By: the user

previousLine
The user pressed the left arrow key (for a horizontal track bar) or the down
arrow key (for a vertical track bar) to decrement the current value by the
amount of the current line increment.

Triggered By: the user

previousPage
The user pressed the PageDown key, or clicked to the left of (for a horizontal
track bar) or below (for a vertical track bar) the slider bar, to decrement the
current value by the amount of the current page increment.

Triggered By: the user

rightClicked
The user clicked mouse button 2 somewhere on the track bar.

Triggered By: the user

tabbed
The user pressed the Tab key while the track bar had the input focus. You
can connect this event to the setFocus message of another part to set the tab
order.

Triggered By: the user

Messages
A track bar has the following messages.

decrementPosition
Decrement the current position by the current line increment.

Returns: —
Triggers: —

decrementPositionBy: anInteger
Decrement the current position by the specified amount.

Returns: —
Triggers: —

disable
Sets the track bar to ignore user input; however, the position can still be
changed with a message.

Returns: —
Triggers: —
enable
Sets the track bar to accept user input.

Returns: —
Triggers: —

incrementPosition
Increment the current position by the current line increment.

Returns: —
Triggers: —

incrementPositionBy: anInteger
Increment the current position by the specified amount.

Returns: —
Triggers: —

linIncrement
Answer the current line increment.

Returns: an Integer
Triggers: —

linIncrement: anInteger
Set the current line increment.

Returns: —
Triggers: —

maximum
Answer the maximum value of the range.

Returns: an Integer
Triggers: —

maximum: anInteger
Sets the maximum value of the range.

Returns: —
Triggers: —

minimum
Answer the minimum value of the range.

Returns: an Integer
Triggers: —

minimum: anInteger
Sets the minimum value of the range.

Returns: —
Triggers: —

minimum: anInteger maximum: anInteger
Set the minimum and maximum values of the range.

Returns: —
Triggers: —
minimum: anInteger
maximum: anInteger
position: anInteger

Set the minimum and maximum values of the range, as well as the initial position.

Returns: —
Triggers: —

pagelIncrement

Answers the current page increment.

Returns: an Integer
Triggers: —

pagelIncrement: anInteger

Set the page increment to the specified positive integer value.

Returns: —
Triggers: —

position

Answers the current position.

Returns: an Integer
Triggers: —

position: anInteger

Sets the initial position to the specified integer value, which must be within the range defined by the minimum and maximum values.

Returns: —
Triggers: —

setValue: anInteger

Sets the current position to the specified integer value, and trigger the changed: event if the specified value is different from the previous value.

Returns: —
Triggers: changed:

value

Answers the current position.

Returns: an Integer
Triggers: —

value: anInteger

Sets the initial position to the specified integer value, which must be within the range defined by the minimum and maximum values.

Returns: —
Triggers: —
Direct-Edit

There is no direct-editing for the track bar. If you attempt to direct-edit this part, you receive the following dialog:

Property Edit

Each of the part-specific controls in the track bar’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Position

A non-negative integer that is the initial position. This value must be within the track bar’s range, as defined by the minimum and maximum values. The default value is 0.
Track Bar

**Minimum**
A non-negative integer (which must be less than the maximum value) that is the minimum value of the track bar’s range. The default value is 0.

**Maximum**
A positive integer (which must be greater than the minimum value) that is the maximum value of the track bar’s range. The default value is 100.

**Line Increment**
A positive integer that is the amount of an individual increment within the track bar’s range. The default value is 1.

**Page Increment**
A positive integer that is the amount of a “paged” increment within the track bar’s range. This is the amount that the slider bar moves when the user clicks on the shaft of the track bar (on either side of the slider bar). The default value is 10.

![TrackBar properties dialog](image)

**Horizontal**
This radio button establishes a horizontal orientation for the track bar, with the minimum value on the left and the maximum value on the right. This is the default orientation.

**Vertical**
This radio button establishes a vertical orientation for the track bar, with the minimum value at the bottom and the maximum value at the top.
Ticks on Left/Top
This radio button causes the tick marks along the shaft of the track bar to be displayed to the left when vertically oriented and on top (above) when horizontally oriented.

Ticks on Both Sides
This radio button causes the tick marks along the shaft of the track bar to be displayed on both sides (left and right when vertically oriented, and above and below when horizontally oriented).

Ticks on Right/Bottom
This radio button causes the tick marks along the shaft of the track bar to be displayed to the right when vertically oriented and on the bottom (below) when horizontally oriented. This is the default value.

Related Parts
- Scroll Bars (Win32 only)
  A scroll bar, either vertical or horizontal, is a visual part that allows you to set an integer value by moving the scroll box along the scroll bar.
- Sliders (OS/2 only)
  A slider control, either vertical or horizontal, is a visual part that lets a user set, display, or modify a value by moving the slider arm along the slider shaft.
This example demonstrates the Track Bar part's capabilities.

This example is in the file RANGEX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the links repositioned for greater clarity.)

For further details about the progress bar part used in this example, see the example in the Progress Bar chapter earlier in this document.
Tree View
(Win32 only)

The tree view is a visual part that displays a hierarchical list of items, with sublevels that can be expanded or collapsed. Options of the tree view (implemented as properties) include:

- Buttons (marked with plus or minus signs) for expanding or collapsing the hierarchy
- Lines connecting the levels of the hierarchy
- Lines drawn from the root level
- Using or ignoring the optional image supplied for each item
- Allowing the user to directly edit the text label for each item

The Windows 95 Explorer uses a tree view in its left pane, as shown in the following illustration:

Operation

You can expand or collapse the hierarchy shown in the tree view by double-clicking on the text label of an item, or by clicking once on its optional button.

If the Direct Edit property is set, you can direct-edit the text label of the currently selected item by clicking on it again. To cancel this renaming operation, press the Escape key. To complete this renaming operation, press the Enter key or select a different item.
Events

A tree view has the following events.

**aboutToChangeLabel:** `anObject to: aString`
Triggered when the user has directly edited the contents of an item, and that change is about to be committed; this is the opportunity for validation to take place. The first event argument specifies the item that was directly edited, and the second argument specifies the new text that was entered. An event handler for this message may send the message `#abortChange` to the control to cancel the direct editing of that item.

Triggered By: the user

**aboutToEditLabel:** `anObject`
Triggered with the list item as an argument when that item was selected for direct editing. An event handler for this message may send the message `#abortChange` to the control to disallow direct editing of that item.

Triggered By: the user

**changed:** `anObject`
The selection was changed to that specified by the event argument.

Triggered By: the user

**changedLabel:** `anObject to: aString`
Triggered after a direct edit by the user was successfully completed. The first event argument is the object which was direct edited, and the second argument specifies the new text.

Triggered By: the user

**collapsed:** `anObject`
Triggered when an expanded item is collapsed by either clicking on its optional button (marked with a minus sign) or by double-clicking the item.

Triggered By: the user

**doubleClicked:** `anObject`
Triggered when an item is double-clicked with mouse button 1; the event argument indicates the list element that was double-clicked.

Triggered By: the user

**expanded:** `anObject`
Triggered when a collapsed item is expanded by either clicking on its optional button (marked with a plus sign) or by double-clicking the item.

Triggered By: the user

**needsChildrenFor:** `anObject`
The control needs the child items for `anObject`. The event handler should answer an Indexed Collection (Array, OrderedCollection, or SortedCollection) of the child objects of the specified list item. If not handled, it is assumed that there are no child items.
needsContents
Occurs when the tree view needs to have its contents (either when it is about
to be opened, or when it has been notified to update its contents). An event
handler would respond with the contents: message.

needsHasChildrenFor: anObject
The control needs to know if anObject has any child items. The event
handler should answer a Boolean indicating whether the specified list item
has any child items. If not handled, the control will determine this from the
answer returned by the event handler in response to the needsChildrenFor:
message.

This event should be handled in situations (such as traversing the file system)
when it is faster to ask “Do you have any children?” than it is to ask “How
many children do you have?”.

needsImageFor: anObject
The control needs the optional image for anObject. The event handler should
return a 16 x 16 bitmap or small icon.

needsSelectedImageFor: anObject
The control needs the “selected” image representation (that is, the image to
be displayed when the item is selected in the list) for anObject. The event
handler should return a 16 x 16 bitmap or small icon.

needsStringFor: anObject
The control needs the displayable string for anObject. The event handler
should return a String. If not handled, the #asString message is used.

tabbed
Occurs when the Tab key is pressed and the control has the input focus. You
can connect this event to the setFocus message of another part to set the tab
order.

Triggered By: the user

Messages

A tree view has the following messages.

collapseItem: anObject
Collapse the children of anObject (the argument).

Returns: —
Triggers: collapsed:

contents
Answer the contents (root level of the hierarchy) of the receiver.

Returns: a Collection of TreeViewItems
Triggers: —

contents: aCollection
Set the contents (root level of the hierarchy) of the receiver to aCollection.

Returns: —
Triggers: —
deleteAll
   Delete the whole list.
   Returns: —
   Triggers: —

deleteItem: anObject
   Delete anObject from the items in the receiver.
   Returns: —
   Triggers: —

disable
   Disable user interaction with the hierarchical list.
   Returns: —
   Triggers: —

enable
   Enable user interaction with the hierarchical list.
   Returns: —
   Triggers: —

expandItem: anObject
   Expand the children of anObject.
   Returns: —
   Triggers: expanded:

selectItem: anObject
   Set the current selection to anObject.
   Returns: —
   Triggers: —

setFocus
   Moves the input focus to the hierarchical list.
   Returns: —
   Triggers: —

**Direct-Edit**

Direct-editing the tree view in the workbench allows you to enter the initial contents. Use the Enter key to separate items, and the Tab key to denote each sublevel in the hierarchy. Click outside the tree view when you are done entering items.

If the Direct Edit property is set, at runtime the user can edit the text label of the selected item (as explained in the Operation topic earlier in this chapter).
Property Edit

Each of the part-specific controls in the tree view’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

Indent (pixels)

This integer entry field specifies the indentation (in pixels) of each level in the hierarchy.

Direct Edit

When this property is checked, the user is allowed to directly edit the text label of an item in the list. By default, direct editing is not allowed.

Lines

When this property is checked, the list includes lines connecting the levels of the hierarchy. By default, these lines are included in the list.

Lines At Root

When this property is checked, the list includes lines drawn from the root level. By default, these lines are included in the list.
Tree View (Win32 only)

Buttons
When this property is checked, the list includes buttons (marked with plus or minus signs) that the user can click on to expand or collapse the hierarchy. When this property is unchecked, the buttons are not present but the user can double-click on items in the list to expand or collapse the hierarchy. By default, the buttons are included in the list.

Use Images
When this property is checked, the small images supplied for the items in the list are displayed. When this property is unchecked, only the text labels are displayed. By default, the supplied images are used.

Related Parts
- **List View (Win32 only)**
  The list view is a visual part that can display a collection of items several different ways: in icon view, small icon view, list view, or report view.

Example
This example demonstrates the Tree View part’s capabilities. It illustrates how to build a user interface similar to that of the Windows 95 Explorer, using the tree view and list view controls. The user can dynamically change the view displayed by the list view, and can direct-edit both the tree view and the list view.

This example is in the file TVLVX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)

For further details about the list view part used in this example, see the example in the List View (Win32 only) chapter earlier in this document.
Tree View (Win32 only)
A value holder is a nonvisual part that serves as a variable to hold a value. Both events and messages produce values that you may need to store for later access.

When a link fires, the event value is available only to links in the same link chain. After the last message in the link chain finishes executing, the event value is no longer accessible. Similarly, message results are available at the time message execution finishes, but they disappear unless they are explicitly stored or are used as argument values for messages later in the chain.

To save an event value or message result for later reference, you need a part that has a message that causes the part to store the value of the message argument, and another message that returns the stored value. In conventional programming languages, variables perform these functions.

In PARTS Workbench, the value holder part and its specializations serve this purpose. The `setValue:` message sets the contents of the value holder to be the argument value. The `value` message retrieves the contents. The `changed:` event triggers whenever a value is stored into the value holder, even if the new value is the same as the previously stored value.

**Operation**

**Typed Value Holders**

One of the properties of a value holder is the class or type of value it contains. Each class of data objects understands a set of messages that define its behavior, just as each different kind of part understands its own set of messages. When you set the class property for a value holder, the messages for that class are added to those of the value holder itself.

Storing an object of the wrong class into a value holder (i.e., sending the message `setValue:` with an argument that is a String to a value holder whose class was set to `Integer`) triggers the `valueException:` event.

The Catalog includes the value holder specializations array holder, dictionary holder, number holder, ordered collection holder, string holder, and string template parts. These are generally equivalent to value holders that have already had their type and initial value properties set appropriately (some of the specializations have additional features which would not be available simply by changing the class of a value holder).

These parts should be used whenever you need a value holder for one of these classes, since their icons visually represent their class in your application and they have special Properties dialogs for initializing their values.
Setting the Class of a Value Holder

You set the value holder part’s class property as follows:

1. Select the value holder in your application.
2. Choose the Select/Edit properties... command.
   The Properties dialog is displayed.
3. Enter the name of the class into the Class: field.
4. Click the OK button to close the Properties dialog.

Once you set the type of an value holder, you can link to any of the messages of that type.

Untyped Value Holders

If you don’t set the class property of the value holder, you can store and retrieve values of different types, but can’t send any class-specific messages directly to the value. If you want to store and retrieve different types of values in the same value holder, do not set the class property.

Initialization

One of the properties of a value holder is its initial value. You specify it by entering its value into the New value: entry field of its Properties dialog. The initial value doesn’t need to be set if you store a value into the value holder with the setValue: message before executing any messages.

The specialized value holders in the Catalog have special properties dialogs that make it easier to enter and modify their initial values.

**TIP:** There are two good ways to ensure that the value holder will be initialized before its value is accessed — the first and only sure-fire method is to set the value directly with the properties dialog. The second way is to link an opened event to a message that sets the value. If you choose the latter, make sure that there aren’t any other chains off the opened event that might access the value before it’s been set.

An initialized value holder is used either as a constant, or as an initialized variable, depending on whether you ever change the value in the value holder.

The value of an uninitialized value holder is **nil**, a special value used as a value when no other object is appropriate.

**To set the initial value for a value holder:**

1. Select a value holder.
2. Choose the Selected/Edit properties... command.
3. In the New value: field, enter a constant or a PARTStalk expression that evaluates to a value of the proper class (if one has been set).
4. Click the OK button to evaluate the expression and close the Properties dialog.
Events

changed: aValue
Occurs when the contents of the value holder are replaced by a different value. (This event is not triggered when the argument of the setValue: message is equal to the current value in the value holder.) The new value is provided by the event value.

Triggered By: setValue:

valueException: aValue
Occurs when a value is stored into a value holder that has its class set and the value is not of the proper type. The event value is the argument to setValue:

Triggered By: setValue:

Messages

The following messages are the core set that every value holder supports. When the class is set, a value holder also supports all the messages that are appropriate for that class.

setValue: aValue
Sets the value holder contents to the argument. If the class has been specified, the argument is checked to make sure that it is of the proper type. If it is not the proper type, the valueException: event is triggered.

Returns: aValue
Triggers: changed:

value
Returns the value contained in the value holder.

Returns: a Value
Triggers: —

valueCopy
Returns a copy of the value in the value holder.

Returns: a Value
Triggers: —

valueIsEqual: aValue
Returns true if aValue is equal to the value in the value holder, otherwise false.

Returns: a Boolean
Triggers: —

valueIsNil: aValue
Returns true if the value in the value holder is nil, otherwise false.

Returns: a Boolean
Triggers: —
**Value Holder**

**valueNotEqual: aValue**

Returns true if aValue is not equal to the value in the value holder, otherwise false.

*Returns:* a Boolean

*Triggers:* —

**valueNotNil: aValue**

Returns true if the value in the value holder is not nil, otherwise false.

*Returns:* a Boolean

*Triggers:* —

**Direct-Edit**

Direct-editing a value holder lets you change the part name that appears beneath the icon in the workbench. It is a good idea to label the variables and constants in your program, just as you would name them in a programming language.

**Property Edit**

Each of the part-specific controls in the value holder’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, *Standard Part Properties*.

![Value Holder Properties Dialog](image)

**Class**

If you want to set the class of the value holder so you can send class-specific messages to the value it contains, enter the class name in this field. When set, only values having the same class can be stored in the value holder. The default is to not set the class, allowing values of any class to be stored.

**Value**

This static text displays the current value held by the value holder. The default value is nil.

**New value**

This is an entry field that lets you type an expression for a value in the same format that would be used in a script.
Edit value...

Opens a dialog that lets you edit the current value.

Related Parts

The following parts are value holders for the most common types of values. Each of these has a customized Properties dialog to let you easily initialize value holders.

- **Array Holder**
  A value holder part with its class set to Array.

- **Dictionary Holder**
  A value holder part with its class set to Dictionary.

- **Number Holder**
  A value holder part with its class set to Number.

- **Ordered Collection Holder**
  A value holder part with its class set to OrderedCollection.

- **String Holder**
  A value holder part with its class set to String.

Example

Following are some examples of how value holders and their specializations can be used. In the first illustration that follows, the String typed into the entry field is used as an argument to the disk accessor’s `setPattern:` message. In order to store the String for later access, a string holder is used to store it first and its `changed:` event is used to send the String to the `setPattern:` message.

![Example Diagram](image-url)
The following produces the same results, but uses a result link to pass along the value since the `setValue:` message returns its argument after it has set the contents of the string holder.

For illustrative purposes, the following application uses two value holder parts: a number holder and a value holder with its class set to Float. Two number holders could have been used instead. The upper one, the number holder, is used as a variable to hold the total value. The value holder is used as a constant to hold the Float value 0.15.

When the user enters a number in the top currency entry field and presses the Calculate 15% tip button, the entry field loses focus, which triggers the `changed:` event and sets the value of the number holder. Triggering the button’s `clicked` event causes the numeric message `*` (multiplication) to be sent to the number holder. This message requires an argument, which is supplied by the value of the value holder. The result of that message supplies the argument value for the `setValue:` for the lower value holder.
After the user types the value 100.00 at runtime and presses the button, the result looks like the following:
A value set part is a visual part that provides a general selection capability. A selection is chosen from a rectangular table of cells, each containing text, an icon, a bitmap, or a color.

Cells are addressed in two different ways – by coordinate or by index.

Each cell has a row and column coordinate represented by the X and Y values of a point. Columns and rows are numbered starting with one in the upper left corner. The following illustration shows the cell coordinates for a table with two rows and three columns:

```
1@1 2@1 3@1
1@2 2@2 3@2
```

Each cell also has an integer index, which is the number of the cell starting with one at the upper left and counting from left to right, top to bottom. The following illustration shows the cell indices for a table with two rows and three columns:

```
1 2 3
4 5 6
```

One of the cells is always selected. The selected cell is indicated by a frame around it. You can change the selection by clicking on another cell or by pressing the arrow keys. Changing the selection triggers the `changedCoordinate:` and `changedIndex:` events.

You can initiate an operation on the selected cell by double clicking over the cell or pressing the Enter key when the cell is selected. These actions trigger the `doubleClickedCoordinate:` and `doubleClickedIndex:` events.

You use the property dialog to set the contents of each cell to one of the following types:

- Bitmap
- Color
- Icon
- Text

The contents of the cells in the value set cannot be changed at application execution time.
Value Set (OS/2 only)

The icon editor dialog uses a value set (in the lower left corner) to select one of sixteen colors.

![Icon Editor Dialog with Value Set](image)

**Operation**

To change the selected cell in the value set part:

> Click on the cell.

or

> Use the arrow keys to move the selection to an adjacent cell.

To initiate an operation on the selected cell:

> Double click on the cell.

or

> Press the Enter key.

**Events**

**aboutToChange**

Occurs when a change of selection is requested by clicking on another cell or pressing an arrow key. If the `abortChange` message is received during the firing of any link connected to this event, the requested selection is canceled and the current selection does not change.

*Triggered By:* the user

**changedCoordinate:** `aPoint`

Occurs when mouse button 1 is clicked over a cell in the value set, or an arrow key is pressed. The event value `aPoint` contains the row and column coordinates of the newly selected cell in the X and Y values.

*Triggered By:* the user

**setValueCoordinate:**

**setValueIndex:**
**ValueChanged:** anInteger

Occurs when the selection changes. The event value anInteger is the index of the cell, starting at one and counting from left to right, top to bottom.

*Triggered By:* the user

**setValueCoordinate:**

**setValueIndex:**

**doubleClickedCoordinate:** aPoint

This event occurs when mouse button 1 is double-clicked over a cell in the value set or the Enter key is pressed. The event value aPoint contains the row and column coordinates of the selected cell. The **changedCoordinate:** event is triggered on the first click of the double-click, the **doubleClickedCoordinate:** event is triggered on the second click.

*Triggered By:* the user

**doubleclickedIndex:** anInteger

This event occurs when mouse button 1 is double clicked over a cell in the value set or the Enter key is pressed. The event value anInteger is the index of the selected cell. The **changedIndex:** event is triggered on the first click of the double-click, the **doubleClickedIndex:** event is triggered on the second click.

*Triggered By:* the user

**rightClicked**

Occurs when mouse button 2 is clicked over the value set.

*Triggered By:* the user

**tabbed**

Occurs when the Tab key is pressed and the value set has the input focus. You can connect this event to the **setFocus** message of another part to set the tab order.

*Triggered By:* the user

**Messages**

**abortChange**

Cancels the selection in progress if it is executed as a consequence of triggering the **aboutToChange** event.

*Returns:* —

*Triggers:* —

**disable**

Disables the value set so that it ignores user input.

*Returns:* —

*Triggers:* —

**enable**

Enables the value set to respond to user input.

*Returns:* —

*Triggers:* —
**Value Set (OS/2 only)**

**setFocus**
Moves the input focus to the value set.

*Returns:* —
*Triggers:* —

**setValueCoordinate:** *aPoint*
Sets the selection to the cell whose address is the argument. If *aPoint* does not address one of the of items in the value set, no item is selected.

*Returns:* *aPoint*
*Triggers:* changedCoordinate:
  changedIndex:

**setValueIndex:** *anInteger*
Sets the selection to the cell whose index is the argument. If *anInteger* does not address one of the of items in the value set, no item is selected. The argument is the index of the cell to be selected.

*Returns:* *anInteger*
*Triggers:* changedCoordinate:
  changedIndex:

**valueCoordinate**
Returns the row and column coordinates of the selected cell.

*Returns:* a Point
*Triggers:* —

**valueIndex**
Returns the index of the selected cell.

*Returns:* an Integer
*Triggers:* —

**Direct-Edit**
Direct editing the value set brings up the Edit dialog described in the following *Property Edit* section.
Property Edit

Each of the part-specific controls in the value set’s Properties dialog is described in the following paragraphs. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

![Value Set Properties](image)

### Rows

The number of rows of cells in the value set.

### Columns

The number of columns of cells in the value set.

### Styles

- **Border**
  
  The border is a single line drawn around the entire value set. It can be turned on or off.

- **Item Border**
  
  The item border is a single line drawn around each item in the value set. It can be turned on or off.
**Value Set (OS/2 only)**

**Edit contents...**

Opens the Edit dialog that lets you set the type and contents of each of the cells in the value set.

![Value Set Properties - Edit dialog](image)

- **Row and Column**
  Select the row and column of the cell to be edited.

- **Item Type**
  Select the type for this cell.

- **Edit Item...**
  Press the Edit Item... button to open an editor appropriate for the type of the cell — either a bit editor, icon editor, color editor, or a text editor.

- **OK and Cancel**
  Press the OK button to accept the changes for each cell. Press the Cancel button to abandon your changes.

**Related Parts**

- **List Pane**
  You can select one choice from a list of text-only choices using a list pane.

- **Radio Button**
  You can also make one choice from many using a group of radio buttons.

**Remarks**

Even though text is supported, the primary purpose of a value set is to display choices as graphical images. By using graphical images in a value set, you can preserve space on the display screen. You can also see exactly what is being selected instead of having to rely on descriptions of the choices. This allows you to make a selection more quickly than if you had to read a description of each choice. For example, if you want to allow a user to choose from a variety of patterns, you can present those patterns as value set choices, instead of having to provide a list of radio buttons with a description of each pattern.
If long strings of data are to be displayed as choices, radio buttons or list panes should be used. However, for small sets of numeric or textual information, either a value set or radio buttons are appropriate.

Example

The example value set has one cell set to each of the four types of value set cells. When a cell is selected, its coordinates and index appear in the two leftmost entry fields. When it is double clicked, its coordinates and index appear in the rightmost entry fields. When double clicking, note that the `changedIndex:` and `changedCoordinate:` events trigger on the first click, and the `doubleClickedIndex:` and `doubleClickedCoordinate:` events trigger on the second click.

A confirmation dialog appears whenever the selection is changed to demonstrate the use of `aboutToChange` and `abortChange`. Choosing `No` in the confirmation dialog aborts the selection.

This example is in the file `VALUESET.PAR` in the `SAMPLE\PARTDEMO` subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
The video pane is a visual part that is used to display AVI (Audio / Video Interleaved) full-motion video files.

The contents of a video pane is an instance of the AviVideo class, which specifies an individual .AVI file. The AVI Video (Win32 only) part in the catalog can be used to set the contents of the video pane.

**Operation**

The display can be scrolled while the video is playing.

**Events**

The video pane has no events.

However, the `needsContents` event occurs when the pane is about to be opened, to request that its contents be set. An event handler would respond with the `#contents:` message. Similarly, the `needsMenu` event occurs when the pane is about to be opened and its menu needs to be added to the menu bar. An event handler would respond with the `#setMenu:` message.

**Messages**

A video pane has the following messages.

- **contents**
  
  Answer the contents of the receiver. (This message is an alias for the `video` message.)

  **Returns:** an AviVideo

  **Triggers:**

- **contents:** aVideo

  Set the contents of the receiver to `aVideo`. (This message is an alias for the `video` message.)

  **Returns:**

  **Triggers:**

- **disable**

  Causes the video pane to ignore user input.

  **Returns:**

  **Triggers:**
enable
   Enables the video pane to respond to user input.
   Returns: —
   Triggers: —

setFocus
   Moves the input focus to the video pane.
   Returns: —
   Triggers: —

video
   Answer the contents of the receiver.
   Returns: an AviVideo
   Triggers: —

video: aVideo
   Set the contents of the receiver to aVideo.
   Returns: —
   Triggers: —

Direct-Edit

There is no direct-editing for the video pane.

Property Edit

The part-specific property of the video pane is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Stretch
This property determines whether the image is sized to fit the current dimensions of the video pane. This property is turned off by default.

Related Parts
- **AVI Video (Win32 only)**
  The AVI video part is a nonvisual part that is used to play AVI video files in a video pane.
Video Pane (Win32 only)

Example

This example demonstrates a visual interface to the Video Pane part's capabilities. The File button opens a dialog where you select the desired file. Other push buttons control the playing of the file, and information about the selected file is displayed. (You could provide a fancier interface by replacing the labeled push buttons with a toolbar having VCR-style bitmap graphics.)

Check boxes determine whether audio and video output are enabled, and whether the video is stretched to fit the video pane. When the Stretch feature is checked, scroll bars in the video pane allow the video to be positioned within the frame of the pane.

This example is in the file VIDEOX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)

For further details about the AVI video part used in this example, see the example in the AVI Video (Win32 only) chapter earlier in this document.
Wave Audio
(Win32 only)

This nonvisual part is used to play wave audio files, which are digital audio samples in .WAV format, using the Media Control Interface. To use this part, you must have a sound card and speakers and an MCI Wave Audio device driver.

Operation

To make effective use of this part’s capabilities, you will probably need to create an application with a visual interface, as shown in the Example topic later in this chapter.

The open message should be the first message sent to this part, and the close message should be the last message sent to it.

Events

A wave audio part has the following event.

playCompleted
Indicates that playing (initiated by a play, playFrom, or playFrom:to: message) has completed. Triggered by the MCI device driver when the end of the media is reached.

Triggered By: MCI device driver

Messages

A wave audio part has the following messages.

bitsPerSample
Answer the number of bits per sample for the current media.

Returns: an Integer
Triggers: —

bytesPerSecond
Answer the number of bytes per second for the current media.

Returns: an Integer
Triggers: —

channels
Answer the number of channels for the current media.

Returns: an Integer
Triggers: —
close
Close the receiver.
Returns: —
Triggers: —

filename
Answer the current filename.
Returns: a String
Triggers: —

filename: aString
Set the current filename to aString.
Returns: —
Triggers: —

isNotReady
Answer whether the device state is 'Not Ready'.
Returns: a Boolean
Triggers: —

isOpen
Answer whether the device has been opened.
Returns: a Boolean
Triggers: —

isPaused
Answer whether the device state is 'Paused'.
Returns: a Boolean
Triggers: —

isPlaying
Answer whether the device state is 'Playing'.
Returns: a Boolean
Triggers: —

isReady
Answer whether the device state is 'Ready'.
Returns: a Boolean
Triggers: —

isSeeking
Answer whether the device state is 'Seeking'.
Returns: a Boolean
Triggers: —

isStopped
Answer whether the device state is 'Stopped'.
Returns: a Boolean
Triggers: —
isTimeFormatBytes
Answer whether the time format is Bytes.
Returns: a Boolean
Triggers: —

isTimeFormatMilliseconds
Answer whether the time format is Milliseconds.
Returns: a Boolean
Triggers: —

isTimeFormatSamples
Answer whether the time format is Samples.
Returns: a Boolean
Triggers: —

length
Answer the length of the receiver, in the current time format.
Returns: a String
Triggers: —

open
Open the receiver.
Returns: —
Triggers: —

pause
Pause the receiver.
Returns: —
Triggers: —

play
Begin playing the receiver starting at the current position.
Returns: —
Triggers: —

playFrom: startFrame
Begin playing the receiver starting at the position indicated by startFrame.
Returns: —
Triggers: —

playFrom: startFrame to: endFrame
Begin playing the receiver starting at the position indicated by startFrame, and stop playing when the position reaches endFrame.
Returns: —
Triggers: —

position
Answer the current position, in units of the current time format.
Returns: a String
Triggers: —
position: anIntegerOrString
Set the current position to anIntegerOrString (in units of the current time format). A String is permitted so that non-integer time formats can be specified.

Returns: —
Triggers: —

reset
Reset the position to the beginning of the file.

Returns: —
Triggers: —

resume
Resume playback from the current position, following a pause.

Returns: —
Triggers: —

samplesPerSecond
Answer the number of samples per second for the current media.

Returns: an Integer
Triggers: —

stop
Stop playing.

Returns: —
Triggers: —

timeFormat
Answer a String which indicates the current time format.

Returns: a String
Triggers: —

timeFormat: aTimeFormatString
Set the current time format to aTimeFormatString.

Returns: —
Triggers: —

timeFormatBytes
Set the time format to Bytes.

Returns: —
Triggers: —

timeFormatMilliseconds
Set the time format to Milliseconds.

Returns: —
Triggers: —

timeFormatSamples
Set the time format to Samples.

Returns: —
Triggers: —
Direct-Edit

Direct-editing the wave audio part lets you change the part name (directly on the workbench).

Property Edit

The part-specific property of the wave audio part is described in the following paragraph. For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.

File Name...

This push button opens a standard file browser so you can locate and select the filename of the wave audio file.

Related Parts

- CD Audio (Win32 only)
  The CD audio part is a nonvisual part that is used to play CD audio discs.
Example

This example demonstrates a visual interface to the Wave Audio part's capabilities. The File button opens a dialog where you select the desired file. Other push buttons control the playing of the file, and information about the selected file is displayed. (You could provide a fancier interface by replacing the labeled push buttons with a toolbar having VCR-style bitmap graphics.)

This example is in the file WAVEX.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory. (The illustration shows a slightly modified version of this file, with the parts and links repositioned for greater clarity.)
Window

A window is a visual part that contains other visual parts.

Windows and dialog windows are the foundation of your user interface. In order for the user to see a part, it must be placed in a window or a dialog window. The most important difference between the two is that dialog windows are modal while windows are non-modal.

For example, if you click on a button in your application that opens a dialog window, you can’t use any other part of your application except for the dialog window until you close the dialog window; this is known as entering a mode. Modal dialogs are useful when you need to prevent the user from returning to the main application until they have finished with the dialog. A confirmer is a good example of a modal dialog: the user must make a decision (Yes/No, OK/Cancel, or Retry/Cancel) before the program can continue executing.

If the button in your application opens a window instead, you can use controls in the new window or any other window in your application without restriction. For example, if your application opens multiple documents, you probably want to open each document in a window rather than a dialog window so that you can switch freely between them.

You can also use windows to create non-modal “dialogs”. The Background Color Settings window (choose File/Settings from the workbench menu bar and press the Background colors... button) is implemented as a window so that you can easily experiment with different background colors. You can change the workbench’s background color, to red for example, and go back to work without closing the Background Color Settings window. If you decide that red isn’t an ideal color for the workbench background, just click on Background Color Settings window and change it; you don’t have to go through the menu commands again.

The Background Color Settings window is independent of the Settings window from which you originally opened it. For example, you can close the Settings window while leaving the Background Color Settings window open. In general, all windows are independent of each other; this is non-modal behavior.

In OS/2, each window may optionally include the following features (which are described in the Property Edit section):

- title bar
- title-bar icon
- menu bar
- minimize button
The maximize button.

In Windows, each window may optionally include the following features (which are described in the Property Edit section):

- title bar
- menu bar
- control-menu box
- minimize button
- maximize button

### Operation

A window can be moved, resized, closed, minimized, and maximized.

#### System Menu (OS/2) or Control Menu (Windows)

**To access the OS/2 system menu:**

> Click on the title-bar icon.

**To access the Windows control menu:**

> Click on the control-menu box.
The choices that appear on the OS/2 system menu or the Windows control menu depend on the specific application and version of the operating system. See your operating system documentation for more information on the choices available on your system.

Minimizing a Window
To shrink the entire window to an icon:
> Click on the Minimize button.
To restore the minimized icon back into a window:
> Double-click on the icon.

Maximizing a Window
To expand the window to fill the entire display:
> Click on the Maximize button.
To restore the window to its original appearance:
> Click on the Maximize button.

Choosing a Menu from the Menu Bar
To select a pull-down menu from the menu bar:
> Click the corresponding keyword.
To make a choice from the menu:
> Click on the desired item.
To cancel the menu without making a choice:
> Move the cursor outside of the menu and click.
   
or
> Press the Escape key.

Moving a Window
1 Move the cursor over the title bar.
2 Drag the window by the title bar to the new location.

Resizing a Window
> Drag the window border to expand or shrink it to the desired size.

Events
aboutToClose
Occurs when the close message is sent or when the user closes the window using the OS/2 system menu or the Windows control menu. If this event does not trigger the execution of the abortClose message, the window closes.

Triggered By: the user
close
Window

**closed**

Occurs just after the window closes.

*Triggered By:* the user

**opened**

Occurs just after the window opens.

*Triggered By:* open

**rightClicked**

Occurs when the user clicks on the window with mouse button 2.

*Triggered By:* the user

**Messages**

**abortClose**

If this message is executed as a consequence of triggering the `aboutToClose` event, the window does not close.

*Returns:* —

*Triggers:* —

**close**

Triggers the `aboutToClose` event. If the message `abortClose` is executed as a result of triggering `aboutToClose`, the window does not close. Otherwise the window triggers the `closed` event and closes.

Note that if the closing window is the last remaining window in the application, the entire application normally ends. To change this default behavior so that the application will continue even in the absence of any window parts, you can execute the following expression:

```
PARTSWindowPart exitOnLastWindow: false
```

*Returns:* —

*Triggers:* `aboutToClose`

**helpManager:** `aHelpManager`

Assigns the “Help Manager” for the window, so you can provide online help for the user. For more information, see the *Help Menu* chapter earlier in this document.

*Returns:* —

*Triggers:* —

**hideWindow**

Makes the window invisible but does not close it.

*Returns:* —

*Triggers:* —

**label**

Returns the window label.

*Returns:* a String

*Triggers:* —
Window

open
Opens up the window and triggers the opened event.

Returns: —
Triggers: opened

setLabel: aString
Sets the title of the window to the argument.

Returns: —
Triggers: —

showWindow
Makes a hidden window visible.

Returns: —
Triggers: —

Direct-Edit
Direct-editing a window lets you enter a new title in the title bar.

Property Edit
Each of the part-specific controls in the window’s Properties dialog is described in the following paragraphs. The OS/2 version of this part has the following additional properties (or choices):

- Dialog border (border style)
- Open as desktop child (opening style)

For an explanation of any of the standard properties, refer to Appendix C, Standard Part Properties.
Window label
This field lets you enter the text that appears in the title bar of the window. The default label of a new window is the title of the application.

End-user help file
This field lets you specify the name of the online help file that will be available to the user when the application is running. For more information about providing online help, see the Help Menu chapter earlier in this document.

Styles
- **Border**
  Controls the appearance of the window border. The two possibilities for Windows are shown in the following illustration:

  ![Size Border](image1)
  ![Single-line Border](image2)

  The three possibilities for OS/2 are shown in the following illustration:

  ![Size Border](image3)
  ![Single-line Border](image4)
  ![Dialog Border](image5)

  If the border style is size-border, the window can be resized by the user. Otherwise the window cannot be resized. The size border is the default.

- **Title bar**
  Controls whether the title bar is displayed at the top of the window. The title bar is displayed by default.

- **Menu bar**
  Controls whether the menu bar and menu choices are displayed below the title bar. The menu bar and menu choices are displayed by default.

- **Control menu (Windows) or System menu (OS/2)**
  Controls whether to display the OS/2 title-bar icon or the Windows control-menu box. The title-bar icon or control-menu box is displayed by default. If you disable the title-bar icon or control menu, make certain that you provide another means of closing the window, such as a *Close* button or a File / Exit menu item connected to the *close* message of the window, otherwise you will not be able to close the window.

- **Minimize button**
  Controls whether to display the minimize button. The minimize button is displayed by default.
• **Maximize button**
  Controls whether to display the maximize button. The maximize button is displayed by default.

**Opening styles...**

This dialog allows the user to choose the opening state of the window part. Note that the OS/2 version of this part has the additional property of Open as desktop child.

![Window Properties - Initial dialog](image)

• **Initial position**
  The Initial Position after opening can be either Relative to the display or Centered on the cursor. Relative to the display uses the position of the window part in relation to the workbench, and during runtime, positions the window in the same relation to the display. Centered on the cursor displays the window at the cursor position when opened. Relative to the display is the default setting.

• **Initial visibility**
  The initial visibility after opening can be either Visible or Hidden. Hidden allows the window to be opened, but not initially visible. In order to be made visible, the `showWindow` message must be sent to the window. If the Visible property is selected, the window will be visible immediately after the `open` message is sent. The `hideWindow` message can be sent to the window to hide it, but not close it. A hidden window retains its current state. Visible is the default setting.

• **Initial size**
  The initial size after opening can be Normal, Maximized, or Minimized. Normal will display the window in the same size as it appears in its expanded state on the workbench. Maximized will display the window in its maximized state on the screen. Minimized will display the window in its minimized state on the screen. Normal is the default setting.

• **Open as desktop child**
  The OS/2 version of this part has the additional property of Open as desktop child. When this property is checked, you can place the dialog window within another window yet have it remain a child of the desktop so you can move it outside of the other window.
Related Parts

- **Dialog Window**
  The dialog window looks like a window, but serves a distinctly different purpose. A window is usually present for the entire time an application is executing. A dialog window is transient, appearing only when the controls it contains are needed.

Example

This example lets you open a window, change the label in the title bar, and close the window. The **opened** and **closed** events are indicated by opening information dialogs when they trigger. When the window is closed, **aboutToClose** brings up a confirmer. If you press No, the **abortClose** message is sent and the window doesn't close.

This example is in the file WINDOW.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.
**Wizard**

*(Win32 only)*

A wizard is a visual part that simulates the Win95 wizard control. A Win95 wizard is a dialog window for guiding the user through the steps of an operation, where typically each step appears on a separate page. Note that the wizard is a modeless dialog.

The wizard part consists of a tab control within a dialog window (labeled “Wizard”) having three push buttons (labeled “Back”, “Next”, and “Cancel”).

The pages of a wizard do not have labeled tabs; instead, the title of the current page is displayed as the title of the wizard’s window.

The following illustration shows how a new wizard part appears in the workbench.

---

**Operation**

Note that the inner area of the tab control belongs to the individual tab control pages. To select or link to the tab control portion of the wizard, you must carefully position the cursor on the border of the tab control (not on its inner area).

**Standard Buttons**

The Back button is disabled when the first page is selected, and the Next button appears as an enabled Finish button when the last page is selected.

- When the Back button is pressed, the `back` event is triggered. If this event is not connected to an action, then the previous page is selected and displayed. Note that a stack is kept of the pages viewed, so that the previous page is the one previously viewed (rather than the physically previous page in the tab control).
• When the Next button is pressed, the **next** event is triggered. If this event is not connected to an action, then the next page is selected and displayed.

• When the Finish button (a special case of the Next button) is pressed, the **finished** event is triggered. This event is prelinked to close the wizard window.

• When the Cancel button is pressed, the wizard window is closed.

**Adding a Help Button**

Because the purpose of a wizard is to help guide a user through a task, you might want to provide additional help information beyond the procedural steps already shown on the page. To do this, you can manually add an optional Help push button to the tab control and link the push button’s **clicked** event to the tab control’s `triggerHelpEvent` message. When the Help button is pressed, the `help` event is triggered for the current page.

For additional information about creating online help for your application, see the `Help Menu` chapter earlier in this document.

**Changing Pages while Building your Wizard**

Because the pages of a wizard do not have labeled tabs, you must perform the following procedure to move between the pages of the tab control when you are building your wizard:

1. Double-click on either the border of the tab control part or on the tab control page, to open the Properties dialog.
2. If the Page page is not currently displayed, click on the Page tab.
3. Click on the desired page to select it, then click the OK or the Apply button.

**Events**

See the `Dialog Window` and `Tab Control (Win32 only)` chapters earlier in this document.

**Messages**

See the `Dialog Window` and `Tab Control (Win32 only)` chapters earlier in this document.

**Direct-Edit**

See the `Dialog Window` and `Tab Control (Win32 only)` chapters earlier in this document.

**Property Edit**

See the `Dialog Window` and `Tab Control (Win32 only)` chapters earlier in this document.
Related Parts

- **Dialog Window**
  The dialog window is a modal pop-up window that can be used for a specific interaction with the user.

- **Tab Control (Win32 only)**
  A tab control is a visual part, similar to a notebook, that holds tab control pages having a labeled tab that can be selected to bring it to the front for viewing.

Example

This example demonstrates the use of the Wizard part for a typical software installation, where the user first enters some registration information and then chooses either a complete installation of all components, a compact installation of predetermined components, or a custom installation of selected components.
When the user chooses a custom installation, a third page is displayed for selecting the components to be installed:

![Wizard screenshot](image)

For a complete or compact installation, this third page is skipped and the fourth page (showing the registration information and the type of installation chosen) is displayed directly. For a custom installation, the fourth page also displays the components selected.

![Wizard screenshot with selected options](image)

This example is in the file WIZARDEM.PAR in the SAMPLE\PARTDEMO subdirectory of your Visual Smalltalk or Visual Smalltalk Enterprise directory.

For additional explanation of the tab control part used in this example, see the Tab Control (Win32 only) chapter earlier in this document.
## Event/Part Cross-Reference

<table>
<thead>
<tr>
<th>Event</th>
<th>Parts having this event</th>
</tr>
</thead>
<tbody>
<tr>
<td>aboutToChange</td>
<td>List Pane, Multiple Choice List, Notebook, Value Set (OS/2 only)</td>
</tr>
<tr>
<td>aboutToClose</td>
<td>Dialog Window, Window</td>
</tr>
<tr>
<td>aboutToUnload</td>
<td>Part Accessor</td>
</tr>
<tr>
<td>callException:arguments:</td>
<td>DLL Accessor</td>
</tr>
<tr>
<td>cancel</td>
<td>Confirmer with Cancel, File Dialog, Prompter</td>
</tr>
<tr>
<td>cellChanged</td>
<td>Table Pane</td>
</tr>
<tr>
<td>cellChangedAtRow:column:value</td>
<td>Table Pane</td>
</tr>
<tr>
<td>cellEnteredAtRow:column:value</td>
<td>Table Pane</td>
</tr>
<tr>
<td>changed:</td>
<td>Alphabetic Entry Field, Array Holder, Combo Box, Currency Entry Field, Date Entry Field, Dial Pane, Dictionary Holder, Drop-down List, Entry Field, Float Entry Field, Graph Pane, Integer Entry Field, List Pane, Multiple Choice List, Number Holder, Ordered Collection Holder, Phone Number Entry Field, Picture Entry Field, Scroll Bar (Win32 only), Slave Spin Button (OS/2 only), Sliders (OS/2 only), Spin Button (OS/2 only), String Holder, Table Pane, Text Pane, Value Holder</td>
</tr>
<tr>
<td>changedCoordinate:</td>
<td>Value Set (OS/2 only)</td>
</tr>
<tr>
<td>changedElement1:</td>
<td>Array Holder</td>
</tr>
<tr>
<td>changedElement2:</td>
<td>Array Holder</td>
</tr>
<tr>
<td>changedIndex:</td>
<td>Drop-down List, List Pane, Value Set (OS/2 only)</td>
</tr>
<tr>
<td>changedPage:</td>
<td>Notebook</td>
</tr>
<tr>
<td>changedPageNumber:</td>
<td>Notebook</td>
</tr>
<tr>
<td>checked</td>
<td>Check Box</td>
</tr>
<tr>
<td>checkedIsTrue</td>
<td>Menu Bitmap Item, Menu Text Item</td>
</tr>
<tr>
<td>clicked</td>
<td>Drawn Button, Menu Bitmap Item, Menu Label, Menu Text Item, Push Button</td>
</tr>
<tr>
<td>clicked:</td>
<td>Check Box, Graph Pane, Radio Button</td>
</tr>
<tr>
<td>closed</td>
<td>Dialog Window, Information Dialog, Window</td>
</tr>
<tr>
<td>columnSelected:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>controlTabbed</td>
<td>Text Pane</td>
</tr>
<tr>
<td>copyException</td>
<td>Clipboard Accessor</td>
</tr>
<tr>
<td>directoryException:</td>
<td>Disk Accessor</td>
</tr>
<tr>
<td>disabledIsTrue</td>
<td>Menu Bitmap Item, Menu Text Item</td>
</tr>
</tbody>
</table>
Appendix A: Event/Part Cross-Reference

doubleClicked: List Pane
doubleClickedCoordinate: Value Set (OS/2 only)
doubleClickedIndex: Value Set (OS/2 only)
doubleClickedOnCell: Table Pane
doubleClickedOnColumn: Table Pane
doubleClickedOnRow: Table Pane
drive: Disk Accessor
doubleClickedOnCell: Table Pane
doubleClickedOnColumn: Table Pane
doubleClickedOnRow: Table Pane
entered: Alphabetic Entry Field, Combo Box, Currency Entry Field, Date Entry Field, Entry Field, Float Entry Field, Integer Entry Field, Phone Number Entry Field, Picture Entry Field
error: Printer
exception: Btrieve Accessor
executeCommand: DDE Server
existsTrue: Disk Accessor, File Accessor
false: Comparison
fileException: File Accessor, Graph Pane, Text Pane
fileNameException: File Accessor
files: Disk Accessor
getException: Clipboard Accessor
gotBitmap: Clipboard Accessor
gotString: Clipboard Accessor
indexException: Array Holder
initiateException:topic: DDE Client
drive: Disk Accessor
keyException: Dictionary Holder
launchException: Launch Pad
loaded: Part Accessor
modifiedIsTrue: Text Pane
no: Confirmer, Confirmer with Cancel
noSuchFileException: File Accessor
ok: Prompter
opened: Dialog Window, Window
openException: DLL Accessor
operationException: Disk Accessor
pathName: Disk Accessor, File Dialog
pokedItem:value: DDE Server
referencedWhileNotLoaded: Part Accessor
<table>
<thead>
<tr>
<th>Event/Part Cross-Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>rightClicked</strong></td>
</tr>
<tr>
<td><strong>rowEntered</strong></td>
</tr>
<tr>
<td><strong>rowSelected</strong></td>
</tr>
<tr>
<td><strong>selected</strong></td>
</tr>
<tr>
<td><strong>selectedIndex</strong></td>
</tr>
<tr>
<td><strong>selectedCell</strong></td>
</tr>
<tr>
<td><strong>selectedGroup</strong></td>
</tr>
<tr>
<td><strong>selectedPoint</strong></td>
</tr>
<tr>
<td><strong>sizeException</strong></td>
</tr>
<tr>
<td><strong>spinnedAround</strong></td>
</tr>
<tr>
<td><strong>subdirectories</strong></td>
</tr>
<tr>
<td><strong>tabbed</strong></td>
</tr>
<tr>
<td><strong>ticked</strong></td>
</tr>
<tr>
<td><strong>tickedTime</strong></td>
</tr>
<tr>
<td><strong>triggered</strong></td>
</tr>
<tr>
<td><strong>triggerFailed</strong></td>
</tr>
<tr>
<td><strong>true</strong></td>
</tr>
<tr>
<td><strong>turnedOn</strong></td>
</tr>
<tr>
<td><strong>unchecked</strong></td>
</tr>
<tr>
<td><strong>unloaded</strong></td>
</tr>
<tr>
<td><strong>valueException</strong></td>
</tr>
<tr>
<td><strong>valuesOn</strong></td>
</tr>
<tr>
<td><strong>yes</strong></td>
</tr>
</tbody>
</table>
## Message/Part Cross-Reference

<table>
<thead>
<tr>
<th>Message</th>
<th>Parts having this message</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Number Holder</td>
</tr>
<tr>
<td>+</td>
<td>Number Holder</td>
</tr>
<tr>
<td>-</td>
<td>Number Holder</td>
</tr>
<tr>
<td>/</td>
<td>Number Holder</td>
</tr>
<tr>
<td>//</td>
<td>Number Holder</td>
</tr>
<tr>
<td>&lt;</td>
<td>Number Holder, String Holder</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Number Holder, String Holder</td>
</tr>
<tr>
<td>&gt;</td>
<td>Number Holder, String Holder</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Number Holder, String Holder</td>
</tr>
<tr>
<td>@</td>
<td>Number Holder</td>
</tr>
<tr>
<td>\</td>
<td>Number Holder</td>
</tr>
<tr>
<td>abortChange</td>
<td>Alphabet Entry Field, Currency Entry Field, Date Entry Field, Drop-down List, Entry Field, Float Entry Field, Integer Entry Field, List Pane, Multiple Choice List, Notebook, Phone Number Entry Field, Picture Entry Field, Value Set (OS/2 only)</td>
</tr>
<tr>
<td>abortClose</td>
<td>Dialog Window, Window</td>
</tr>
<tr>
<td>abs</td>
<td>Number Holder</td>
</tr>
<tr>
<td>abs:</td>
<td>Computation</td>
</tr>
<tr>
<td>add:</td>
<td>Currency Entry Field, Float Entry Field, Integer Entry Field, Ordered Collection Holder</td>
</tr>
<tr>
<td>add:after:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>add:afterIndex:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>add:before:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>add:beforeIndex:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>addAll:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>addFirst:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>addLast:</td>
<td>Ordered Collection Holder</td>
</tr>
<tr>
<td>alarm</td>
<td>Speaker</td>
</tr>
<tr>
<td>append:</td>
<td>Text Pane</td>
</tr>
<tr>
<td>appendToValue:</td>
<td>String Holder</td>
</tr>
<tr>
<td>arcCos</td>
<td>Number Holder</td>
</tr>
</tbody>
</table>
### Appendix B: Message/Part Cross-Reference

<table>
<thead>
<tr>
<th>Method</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>arcSin</td>
<td>Number Holder</td>
</tr>
<tr>
<td>arcTan</td>
<td>Number Holder</td>
</tr>
<tr>
<td>asArray</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>asArray:</td>
<td>Conversion</td>
</tr>
<tr>
<td>asArrayOfSubstrings</td>
<td>String Holder</td>
</tr>
<tr>
<td>asAsciiZ</td>
<td>String Holder</td>
</tr>
<tr>
<td>asCharacter</td>
<td>Number Holder</td>
</tr>
<tr>
<td>asDate</td>
<td>String Holder</td>
</tr>
<tr>
<td>asDate:</td>
<td>Conversion</td>
</tr>
<tr>
<td>asFloat</td>
<td>Number Holder, String Holder</td>
</tr>
<tr>
<td>asFloat:</td>
<td>Conversion</td>
</tr>
<tr>
<td>asInteger</td>
<td>String Holder</td>
</tr>
<tr>
<td>asInteger:</td>
<td>Conversion</td>
</tr>
<tr>
<td>asLowerCase</td>
<td>String Holder</td>
</tr>
<tr>
<td>asLowerCase:</td>
<td>Conversion</td>
</tr>
<tr>
<td>asNumber</td>
<td>Conversion</td>
</tr>
<tr>
<td>asOrderedCollection</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>asParameter</td>
<td>String Holder</td>
</tr>
<tr>
<td>asSet</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>asSortedCollection</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>asString</td>
<td>String Holder</td>
</tr>
<tr>
<td>asString:</td>
<td>Conversion</td>
</tr>
<tr>
<td>asSymbol</td>
<td>String Holder</td>
</tr>
<tr>
<td>asUpperCase</td>
<td>String Holder</td>
</tr>
<tr>
<td>asUpperCase:</td>
<td>Conversion</td>
</tr>
<tr>
<td>at:</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>at:put:</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>beginJob</td>
<td>Printer</td>
</tr>
<tr>
<td>between:and:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>bitmap</td>
<td>Drawn Button, Graph Pane, Menu Bitmap Item</td>
</tr>
<tr>
<td>ceiling</td>
<td>Number Holder</td>
</tr>
<tr>
<td>cellAtName:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>cellAtPoint:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>cellAtRow:column:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>cellAtRow:column:put:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>changeValueBy:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>check</td>
<td>Check Box, Menu Bitmap Item, Menu Text Item</td>
</tr>
<tr>
<td>Word</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>clear</td>
<td>Alphabetic Entry Field, Combo Box, Currency Entry Field, Date Entry Field,</td>
</tr>
<tr>
<td></td>
<td>Entry Field, Float Entry Field, Integer Entry Field, Phone Number Entry</td>
</tr>
<tr>
<td></td>
<td>Field, Picture Entry Field, Table Pane, Text Pane</td>
</tr>
<tr>
<td>clearBitmap</td>
<td>Graph Pane</td>
</tr>
<tr>
<td>clearModified</td>
<td>Text Pane</td>
</tr>
<tr>
<td>clearSelection</td>
<td>Text Pane</td>
</tr>
<tr>
<td>close</td>
<td>Btrieve Accessor, Dialog Window, DLL Accessor, Window</td>
</tr>
<tr>
<td>columnAt:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>columnAt:put:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>concatenate:</td>
<td>Array Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>copy:to:</td>
<td>File Accessor</td>
</tr>
<tr>
<td>copyBitmap:</td>
<td>Clipboard Accessor</td>
</tr>
<tr>
<td>copyBitmapFromScreen</td>
<td>Clipboard Accessor</td>
</tr>
<tr>
<td>copyFrom:to:</td>
<td>Array holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>copyReplaceFrom:to:with</td>
<td>Array holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>copySelection</td>
<td>Text Pane</td>
</tr>
<tr>
<td>copyString:</td>
<td>Clipboard Accessor</td>
</tr>
<tr>
<td>copyToClipboard</td>
<td>Graph Pane</td>
</tr>
<tr>
<td>cos</td>
<td>Number Holder</td>
</tr>
<tr>
<td>create:</td>
<td>Disk Accessor</td>
</tr>
<tr>
<td>currentDate</td>
<td>Timer</td>
</tr>
<tr>
<td>currentRowBottom</td>
<td>Table Pane</td>
</tr>
<tr>
<td>currentRowDown</td>
<td>Table Pane</td>
</tr>
<tr>
<td>currentRowTop</td>
<td>Table Pane</td>
</tr>
<tr>
<td>currentRowUp</td>
<td>Table Pane</td>
</tr>
<tr>
<td>currentTime</td>
<td>Timer</td>
</tr>
<tr>
<td>cutSelection</td>
<td>Text Pane</td>
</tr>
<tr>
<td>date:</td>
<td>File Accessor</td>
</tr>
<tr>
<td>dateTimeString:</td>
<td>File Accessor</td>
</tr>
<tr>
<td>decrementValue</td>
<td>Number Holder</td>
</tr>
<tr>
<td>defaultMessage</td>
<td>Prompter</td>
</tr>
<tr>
<td>defaultPrinterName</td>
<td>Printer</td>
</tr>
<tr>
<td>degreesToRadians</td>
<td>Number Holder</td>
</tr>
<tr>
<td>delete</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>delete:</td>
<td>Disk Accessor, File Accessor</td>
</tr>
<tr>
<td>deleteAll</td>
<td>Combo Box, Drop-down List, List Pane, Multiple Choice List</td>
</tr>
<tr>
<td>deleteItem:</td>
<td>Combo Box, Drop-down List, List Pane, Multiple Choice List</td>
</tr>
<tr>
<td>deleteKey</td>
<td>Text Pane</td>
</tr>
<tr>
<td>denominator</td>
<td>Number Holder</td>
</tr>
</tbody>
</table>

Visual Smalltalk Enterprise Part Reference 621
<table>
<thead>
<tr>
<th><strong>descriptionString:</strong></th>
<th>File Accessor</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>deselectAll:</strong></td>
<td>Multiple Choice List</td>
</tr>
<tr>
<td><strong>deselectIndex:</strong></td>
<td>Multiple Choice List</td>
</tr>
<tr>
<td><strong>deselectItem:</strong></td>
<td>Multiple Choice List</td>
</tr>
<tr>
<td><strong>difference:</strong></td>
<td>Computation</td>
</tr>
<tr>
<td><strong>directoryName:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>directoryPath:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>disable:</strong></td>
<td>Alphabetic Entry Field, Check Box, Combo Box, Currency Entry Field, Date Entry Field, Dial Pane, Drawn Button, Drop-down List, Entry Field, Float Entry Field, Graph Pane, Integer Entry Field, List Pane, Menu Bitmap Item, Menu Text Item, Multiple Choice List, Notebook, Phone Number Entry Field, Picture Entry Field, Push Button, Radio Button, Scroll Bar (Win32 only), Slave Spin Button (OS/2 only), Sliders (OS/2 only), Spin Button (OS/2 only), Static Text, Table Pane, Text Pane, Value Set (OS/2 only)</td>
</tr>
<tr>
<td><strong>divide:</strong></td>
<td>Currency Entry Field, Float Entry Field, Integer Entry Field</td>
</tr>
<tr>
<td><strong>down:</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>driveName:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>element1:</strong></td>
<td>Array Holder</td>
</tr>
<tr>
<td><strong>element2:</strong></td>
<td>Array Holder</td>
</tr>
<tr>
<td><strong>enable:</strong></td>
<td>Alphabetic Entry Field, Check Box, Combo Box, Currency Entry Field, Date Entry Field, Dial Pane, Drawn Button, Drop-down List, Entry Field, Float Entry Field, Graph Pane, Integer Entry Field, List Pane, Menu Bitmap Item, Menu Text Item, Multiple Choice List, Notebook, Phone Number Entry Field, Picture Entry Field, Push Button, Radio Button, Scroll Bar (Win32 only), Slave Spin Button (OS/2 only), Sliders (OS/2 only), Spin Button (OS/2 only), Static Text, Table Pane, Text Pane, Value Set (OS/2 only)</td>
</tr>
<tr>
<td><strong>endJob:</strong></td>
<td>Printer</td>
</tr>
<tr>
<td><strong>even:</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>exceptionString:</strong></td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td><strong>executeCommand:</strong></td>
<td>DDE Client</td>
</tr>
<tr>
<td><strong>executionFailed:</strong></td>
<td>DDE Server</td>
</tr>
<tr>
<td><strong>exists:</strong></td>
<td>Disk Accessor, File Accessor</td>
</tr>
<tr>
<td><strong>exp:</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>factorial:</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>fileExtension:</strong></td>
<td>File Accessor</td>
</tr>
<tr>
<td><strong>fileName:</strong></td>
<td>File Accessor</td>
</tr>
<tr>
<td><strong>files:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>filesIn:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>filesIn:matching:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>filesMatching:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>fileText:</strong></td>
<td>File Accessor</td>
</tr>
<tr>
<td><strong>fileTextToThreshold:</strong></td>
<td>File Accessor</td>
</tr>
<tr>
<td>Message/Part</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>first</td>
<td>Array Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>firstPage</td>
<td>Notebook</td>
</tr>
<tr>
<td>floor</td>
<td>Number Holder</td>
</tr>
<tr>
<td>formFeed</td>
<td>Printer</td>
</tr>
<tr>
<td>frequency:duration</td>
<td>Speaker</td>
</tr>
<tr>
<td>gcd:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>getBitmap</td>
<td>Clipboard Accessor</td>
</tr>
<tr>
<td>getEqual</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getFirst</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getGreaterOrEqual</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getLast</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getLessOrEqual</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getNext</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getPrevious</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>getString</td>
<td>Clipboard Accessor</td>
</tr>
<tr>
<td>hideWindow</td>
<td>Dialog Window, Window</td>
</tr>
<tr>
<td>higherOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>includes:</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>includesKey:</td>
<td>Dictionary Holder</td>
</tr>
<tr>
<td>incrementValue</td>
<td>Number Holder</td>
</tr>
<tr>
<td>initiate</td>
<td>DDE Client, DDE Server</td>
</tr>
<tr>
<td>insert</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>insertItem:</td>
<td>Text Pane</td>
</tr>
<tr>
<td>is:equalTo:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:greaterThan:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:greaterThanOrEqualTo:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:identicalTo:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:lessThan:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:lessThanOrEqualTo:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:notEqualTo:</td>
<td>Comparison</td>
</tr>
<tr>
<td>is:notIdenticalTo:</td>
<td>Comparison</td>
</tr>
<tr>
<td>isAbove:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isAboveEqual:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isBelow:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isBelowEqual:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>is:Checked</td>
<td>Menu Bitmap Item, Menu Text Item</td>
</tr>
<tr>
<td>is:Disabled</td>
<td>Menu Bitmap Item, Menu Text Item</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>isEmpty</td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>isEqualOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isEqualOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isLeftOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isModified</td>
<td>Text Pane</td>
</tr>
<tr>
<td>isRightEqualOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>isRightOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>keyAtValue:</td>
<td>Dictionary Holder</td>
</tr>
<tr>
<td>keys</td>
<td>Dictionary Holder</td>
</tr>
<tr>
<td>keysSorted</td>
<td>Dictionary Holder</td>
</tr>
<tr>
<td>knownDrives</td>
<td>Disk Accessor</td>
</tr>
<tr>
<td>label</td>
<td>Check Box, Dialog Window, Group Pane, Menu Label, Menu Text Item, Push Button, Radio Button, Window</td>
</tr>
<tr>
<td>last</td>
<td>Array Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td>lastPage</td>
<td>Notebook</td>
</tr>
<tr>
<td>launch</td>
<td>Launch Pad</td>
</tr>
<tr>
<td>launch:</td>
<td>Launch Pad</td>
</tr>
<tr>
<td>launchPartFile:</td>
<td>Launch Pad</td>
</tr>
<tr>
<td>lcm:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>left:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>leftMost:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>list</td>
<td>Combo Box, Drop-down List, List Pane, Multiple Choice List</td>
</tr>
<tr>
<td>listValue</td>
<td>Combo Box</td>
</tr>
<tr>
<td>ln</td>
<td>Number Holder</td>
</tr>
<tr>
<td>load</td>
<td>Part Accessor</td>
</tr>
<tr>
<td>lock</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>log:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>logicalAnd:and:</td>
<td>Computation</td>
</tr>
<tr>
<td>logicalOr:and:</td>
<td>Computation</td>
</tr>
<tr>
<td>lowerOf:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>matrixIn:</td>
<td>Table Pane</td>
</tr>
<tr>
<td>max:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>min:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>modulo:and:</td>
<td>Computation</td>
</tr>
<tr>
<td>multiply:</td>
<td>Currency Entry field, Float Entry Field, Integer Entry Field</td>
</tr>
<tr>
<td>negated</td>
<td>Number Holder</td>
</tr>
<tr>
<td>negative</td>
<td>Number Holder</td>
</tr>
<tr>
<td>nextPage</td>
<td>Notebook</td>
</tr>
<tr>
<td>not:</td>
<td>Computation</td>
</tr>
<tr>
<td><strong>numerator</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td><strong>occurrencesOf:</strong></td>
<td>Array Holder, Dictionary Holder, Ordered Collection Holder, String Holder</td>
</tr>
<tr>
<td><strong>odd</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>open</strong></td>
<td>Btrieve Accessor, Confirmer, Confirmer with Cancel, Dialog Window, DLL Accessor, Information Dialog, Nested Part, Window</td>
</tr>
<tr>
<td><strong>openOn:</strong></td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td><strong>openText:</strong></td>
<td>Confirmer, Confirmer with Cancel, Information Dialog</td>
</tr>
<tr>
<td><strong>openTitle:</strong></td>
<td>Confirmer, Confirmer with Cancel, Information Dialog</td>
</tr>
<tr>
<td><strong>page</strong></td>
<td>Notebook</td>
</tr>
<tr>
<td><strong>pageNumber</strong></td>
<td>Notebook</td>
</tr>
<tr>
<td><strong>pasteFromClipboard</strong></td>
<td>Graph Pane</td>
</tr>
<tr>
<td><strong>pasteSelection</strong></td>
<td>Text Pane</td>
</tr>
<tr>
<td><strong>pathName</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>pathNameForOpen</strong></td>
<td>File Dialog</td>
</tr>
<tr>
<td><strong>pathNameForOpen:</strong></td>
<td>File Dialog</td>
</tr>
<tr>
<td><strong>pathNameForSave</strong></td>
<td>File Dialog</td>
</tr>
<tr>
<td><strong>pathNameForSave:</strong></td>
<td>File Dialog</td>
</tr>
<tr>
<td><strong>pathNameOfFile:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>pathNameOfFile:</strong></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><strong>pattern</strong></td>
<td>Disk Accessor, File Dialog</td>
</tr>
<tr>
<td><strong>pen</strong></td>
<td>Graph Pane</td>
</tr>
<tr>
<td><strong>pokeFailed</strong></td>
<td>DDE Server</td>
</tr>
<tr>
<td><strong>pokeItem:</strong></td>
<td>DDE Client</td>
</tr>
<tr>
<td><strong>popUp</strong></td>
<td>Menu</td>
</tr>
<tr>
<td><strong>positive</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>previousPage</strong></td>
<td>Notebook</td>
</tr>
<tr>
<td><strong>print:</strong></td>
<td>Printer</td>
</tr>
<tr>
<td><strong>printerName</strong></td>
<td>Printer</td>
</tr>
<tr>
<td><strong>printerNames</strong></td>
<td>Printer</td>
</tr>
<tr>
<td><strong>printScreen</strong></td>
<td>Printer</td>
</tr>
<tr>
<td><strong>printUserRectangle</strong></td>
<td>Printer</td>
</tr>
<tr>
<td><strong>product:</strong></td>
<td>Computation</td>
</tr>
<tr>
<td><strong>prompt</strong></td>
<td>Prompter</td>
</tr>
<tr>
<td><strong>prompt:default:</strong></td>
<td>Prompter</td>
</tr>
<tr>
<td><strong>promptDefault:</strong></td>
<td>Prompter</td>
</tr>
<tr>
<td><strong>promptMessage</strong></td>
<td>Prompter</td>
</tr>
<tr>
<td><strong>quo:</strong></td>
<td>Number Holder</td>
</tr>
<tr>
<td><strong>quotient:</strong></td>
<td>Computation</td>
</tr>
</tbody>
</table>
Appendix B: Message/Part Cross-Reference

- **radiansToDegrees**: Number Holder
- **raisedTo**: Number Holder
- **raisedToInteger**: Number Holder
- **rem**: Number Holder
- **remainder:and**: Computation
- **remove**: Ordered Collection Holder
- **removeAll**: Dictionary Holder, Ordered Collection Holder
- **removeAll**: Ordered Collection Holder
- **removeBlanks**: Prompter
- **removeFirst**: Ordered Collection Holder
- **removeKey**: Dictionary Holder
- **removeLast**: Ordered Collection Holder
- **rename:to**: File Accessor
- **replaceControlChars**: String Holder
- **replaceControlCharsIn**: Conversion
- **requestItem**: DDE Client
- **reversed**: Array Holder, Ordered Collection Holder, String Holder
- **right**: Number Holder
- **rightMost**: Number Holder
- **ring**: Speaker
- **rounded**: Number Holder
- **roundTo**: Number Holder
- **rowAt**: Table Pane
- **rowAt:put**: Table Pane
- **saveToFile**: Text Pane
- **selectAFont**: Printer
- **selectAll**: Alphabetic Entry Field, Currency Entry Field, Date Entry Field, Entry Field, Float Entry Field, Integer Entry Field, Phone Number Entry Field, Picture Entry Field, Text Pane
- **selectAPrinter**: Printer
- **selectedCell**: Table Pane
- **selectedGroup**: Table Pane
- **selectedText**: Text Pane
- **selectIndex**: Multiple Choice List
- **selectItem**: Multiple Choice List
- **setBitmap**: Drawn Button, Graph Pane, Menu Bitmap Item
- **setColumnLabels**: Table Pane
- **setColumnNames**: Table Pane
- **setColumnWidths**: Table Pane
### Appendix B: Message/Part Cross-Reference

<table>
<thead>
<tr>
<th>Method</th>
<th>Class/Part</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setCommand:</code></td>
<td>Launch Pad</td>
</tr>
<tr>
<td><code>setDefaultMessage:</code></td>
<td>Prompter</td>
</tr>
<tr>
<td><code>setDirectory:</code></td>
<td>Disk Accessor</td>
</tr>
<tr>
<td><code>setDuration:</code></td>
<td>Speaker</td>
</tr>
<tr>
<td><code>setElement1:</code></td>
<td>Array Holder</td>
</tr>
<tr>
<td><code>setElement2:</code></td>
<td>Array Holder</td>
</tr>
<tr>
<td><code>setText:</code></td>
<td>File Accessor</td>
</tr>
<tr>
<td><code>setFileText:to:</code></td>
<td></td>
</tr>
<tr>
<td><code>setFocus</code></td>
<td>Alphabetic Entry Field, Check Box, Combo Box, Currency Entry Field, Date Entry Field, Dial Pane, Drawn Button, Drop-down List, Entry Field, Float Entry Field, Graph Pane, Integer Entry Field, List Pane, Multiple Choice List, Notebook, Phone Number Entry Field, Picture Entry Field, Push Button, Radio Button, Scroll Bar (Win32 only), Slave Spin Button (OS/2 only), Sliders (OS/2 only), Spin Button (OS/2 only), Table Pane, Text Pane, Value Set (OS/2 only)</td>
</tr>
<tr>
<td><code>setFocusAtRow:column:</code></td>
<td>Table Pane</td>
</tr>
<tr>
<td><code>setFrequency:</code></td>
<td>Speaker</td>
</tr>
<tr>
<td><code>setFromBitmapFile:</code></td>
<td>Graph Pane</td>
</tr>
<tr>
<td><code>setFromFile:</code></td>
<td>Text Pane</td>
</tr>
<tr>
<td><code>setFromIconFile:</code></td>
<td>Graph Pane</td>
</tr>
<tr>
<td><code>setFromReference:</code></td>
<td>C Structure, String Holder</td>
</tr>
<tr>
<td><code>setInterval:</code></td>
<td>Timer</td>
</tr>
<tr>
<td><code>setLabel:</code></td>
<td>Check Box, Dialog Window, Group Pane, Menu Label, Menu Text Item, Push Button, Radio Button, Window</td>
</tr>
<tr>
<td><code>setList:</code></td>
<td>Combo Box, Drop-down List, List Pane, Multiple Choice List, Slave Spin Button (OS/2 only), Spin Button (OS/2 only)</td>
</tr>
<tr>
<td><code>setModified</code></td>
<td>Text Pane</td>
</tr>
<tr>
<td><code>setNumberOfColumns:</code></td>
<td>Table Pane</td>
</tr>
<tr>
<td><code>setNumberOfRows:</code></td>
<td>Table Pane</td>
</tr>
<tr>
<td><code>setPattern:</code></td>
<td>Disk Accessor, File Accessor</td>
</tr>
<tr>
<td><code>setPrinterName:</code></td>
<td>Printer</td>
</tr>
<tr>
<td><code>setPromptMessage:</code></td>
<td>Prompter</td>
</tr>
<tr>
<td><code>setRemoveBlanks:</code></td>
<td>Prompter</td>
</tr>
<tr>
<td><code>setRowLabels:</code></td>
<td>Table Pane</td>
</tr>
<tr>
<td><code>setRowNames:</code></td>
<td>Table Pane</td>
</tr>
<tr>
<td><code>setTemplate:</code></td>
<td>String Template</td>
</tr>
<tr>
<td><code>setText:</code></td>
<td>Confirmer, Confirmer with Cancel, Information Dialog</td>
</tr>
<tr>
<td><code>setTitle:</code></td>
<td>Confirmer, Confirmer with Cancel, Information Dialog</td>
</tr>
</tbody>
</table>
Appendix B: Message/Part Cross-Reference

setValue: Alphabetic Entry Field, Array Holder, Check Box, Combo Box, Currency Entry Field, Date Entry Field, Dial Pane, Dictionary Holder, Drop-down List, Entry Field, Float Entry Field, Integer Entry Field, List Pane, Multiple Choice List, Number Holder, Ordered Collection Holder, Phone Number Entry Field, Picture Entry Field, Radio Button, Scroll Bar (Win32 only), Slave Spin Button (OS/2 only), Sliders (OS/2 only), Spin Button (OS/2 only), Static Graphic, Static Text, String Holder, Table Pane, Text Pane, Value Holder

setValueCoordinate: Value Set (OS/2 only)

setValueIndex: Drop-down List, List Pane, Value Set (OS/2 only)

setValueIndices: Multiple Choice List

setWaitForAnswer: Dialog Window, File Dialog, Prompter

showWindow: Dialog Window, Window

sign: Number Holder

sin: Number Holder

tan: Number Holder

sortedAscending: Conversion

sortedDescending: Conversion

spin: Slave Spin Button (OS/2 only), Spin Button (OS/2 only)

sqrt: Number Holder

startTimer: Timer

stopTimer: Timer

subdirectories: Disk Accessor

subdirectoriesIn: Disk Accessor

subtract: Currency Entry Field, Float Entry Field, Integer Entry Field

sum:and: Computation

tan: Number Holder

template: String Template

terminate: DDE Client, DDE Server

text: Confirmer, Confirmer with Cancel, Information Dialog

time: File Accessor

title: Confirmer, Confirmer with Cancel, Information Dialog

tone: Speaker

trigger: Link Junction

trigger: Link Junction

triggerIfEmpty: Link Junction

triggerIfFalse: Link Junction

triggerIfNil: Link Junction

triggerIfNotEmpty: Link Junction

triggerIfNotNil: Link Junction

triggerIfNotNil: Link Junction

628 Visual Smalltalk Enterprise Part Reference
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>triggerIfTrue:</td>
<td>Link Junction</td>
</tr>
<tr>
<td>trimBlanks</td>
<td>String Holder</td>
</tr>
<tr>
<td>truncated</td>
<td>Number Holder</td>
</tr>
<tr>
<td>truncateTo:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>turnOff</td>
<td>Radio Button</td>
</tr>
<tr>
<td>turnOn</td>
<td>Radio Button</td>
</tr>
<tr>
<td>turnToPage:</td>
<td>Notebook</td>
</tr>
<tr>
<td>turnToPageNumber:</td>
<td>Notebook</td>
</tr>
<tr>
<td>uncheck</td>
<td>Check Box, Menu Bitmap Item, Menu Text Item</td>
</tr>
<tr>
<td>undo</td>
<td>Text Pane</td>
</tr>
<tr>
<td>unload</td>
<td>Part Accessor</td>
</tr>
<tr>
<td>unlock</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>up:</td>
<td>Number Holder</td>
</tr>
<tr>
<td>update</td>
<td>Btrieve Accessor</td>
</tr>
<tr>
<td>updateValue:</td>
<td>Array Holder, Dictionary Holder</td>
</tr>
<tr>
<td>value</td>
<td>Alphabetic Entry Field, Array Holder, Check Box, Combo Box, C Structure,</td>
</tr>
<tr>
<td></td>
<td>Currency Entry Field, Date Entry Field, Dial Pane, Dictionary Holder, Drop-</td>
</tr>
<tr>
<td></td>
<td>down List, Entry Field, Float Entry Field, Integer Entry Field, List Pane,</td>
</tr>
<tr>
<td></td>
<td>Multiple Choice List, Number Holder, Ordered Collection Holder, Phone</td>
</tr>
<tr>
<td></td>
<td>Number Entry Field, Picture Entry Field, Radio Button, Scroll Bar (Win32</td>
</tr>
<tr>
<td></td>
<td>only), Slave Spin Button (OS/2 only), Spin Button (OS/2 only), Sliders (OS/2</td>
</tr>
<tr>
<td></td>
<td>only), Static Graphic, Static Text, String Holder, String Template, Table</td>
</tr>
<tr>
<td></td>
<td>Pane, Text Pane, Value Holder</td>
</tr>
<tr>
<td>valueCoordinate</td>
<td>Value Set (OS/2 only)</td>
</tr>
<tr>
<td>valueCopy</td>
<td>Array Holder, Dictionary Holder, Number Holder, Ordered Collection Holder,</td>
</tr>
<tr>
<td></td>
<td>String Holder, Value Holder</td>
</tr>
<tr>
<td>valueFormatted</td>
<td>Alphabetic Entry Field, Currency Entry Field, Date Entry Field, Float Entry</td>
</tr>
<tr>
<td></td>
<td>Field, Integer Entry Field, Phone Number Entry Field, Picture Entry Field</td>
</tr>
<tr>
<td>valueIndex</td>
<td>Drop-down List, List Pane, Value Set (OS/2 only)</td>
</tr>
<tr>
<td>valueIndices</td>
<td>Multiple Choice List</td>
</tr>
<tr>
<td>valuesEqual:</td>
<td>Array Holder, Dictionary Holder, Number Holder, Ordered Collection Holder,</td>
</tr>
<tr>
<td></td>
<td>String Holder, Value Holder</td>
</tr>
<tr>
<td>valuesNil</td>
<td>Array Holder, Dictionary Holder, Number Holder, Ordered Collection Holder,</td>
</tr>
<tr>
<td></td>
<td>String Holder, Value Holder</td>
</tr>
<tr>
<td>valueNotEqual:</td>
<td>Array Holder, Dictionary Holder, Number Holder, Ordered Collection Holder,</td>
</tr>
<tr>
<td></td>
<td>String Holder, Value Holder</td>
</tr>
<tr>
<td>valueNotNil</td>
<td>Array Holder, Dictionary Holder, Number Holder, Ordered Collection Holder,</td>
</tr>
<tr>
<td></td>
<td>String Holder, Value Holder</td>
</tr>
<tr>
<td>write</td>
<td>Btrieve Accessor</td>
</tr>
</tbody>
</table>
This appendix explains the following properties which are common to many parts:

- Backcolor
- Edit bitmap
- Edit icon
- Font
- Forecolor
- Order children
- Part name
- Size window

Any other properties of a particular part are explained in the Property Edit topic within the chapter for that part.

Each part’s properties are displayed in its Properties dialog. You use this dialog to view and change a part’s properties when you are building an application. Some properties are visible as soon as you open the Properties dialog. Others are accessed by pushing a button in the Properties dialog which opens a sub-dialog, as shown in the following illustration.

- Each part has a name property.
- Most visual parts have properties for size, forecolor, backcolor, and font.
- Nonvisual parts and some visual parts have an icon property that allows you to change their visual representation with an icon editor.
- Graphical items like drawn buttons and graph panes have a graphic property that lets you change their appearance with a Bit or Icon editor.
NOTE: Each part Properties dialog also has an Interface... button which opens the part’s Interface window. This is provided as an alternate for the Selected/Edit interface... command available from the Workbench menu bar. A part’s Interface is not one of its properties, however. Closing a Properties dialog, clicking on its OK or Cancel buttons, and undoing property changes have no effect on a part’s Interface window or changes made to the part’s interface.

Backcolor Property

The Backcolor property controls the background color of visual parts. Some visual parts, such as push buttons, don’t have color properties. Pressing the Backcolor... button (or the Forecolor... button) in a visual part’s properties dialog invokes a color dialog.

In Windows, the following standard color dialog appears:

To select a new color in Windows:
1. Double-click on a visual part to open its Properties dialog.
2. Click on the Forecolor... or Backcolor... button.
3. Click on the desired color from the Basic Colors: or Custom Colors: palette and click on the OK button.
4. In the Properties dialog, click on the OK button.

See your Windows documentation for more information on using this Color dialog.
Appendix C: Standard Part Properties

In OS/2, the following color dialog appears:

![Color Dialog](image)

**To select a new color in OS/2:**

1. Double-click on a visual part to open its Properties dialog.
2. Click on the Forecolor... or the Backcolor... button.
3. Choose a system color from the list pane, or click on the RGB Color radio button and mix a color with the R, G, and B sliders.
   
   RGB colors are displayed in a rectangle above the sliders.

4. In the Properties dialog, click on the OK button.

You can set the forecolor and backcolor for an entire group of parts at one time. See **Setting Properties for a Group of Parts** in chapter 4 of the *Visual Smalltalk Enterprise Workbench User's Guide* for information on this procedure.
Edit bitmap Property

The Bit Editor allows you to edit the bitmap displayed by graphical parts such as a menu bitmap item, drawn button, or graph pane.

Press the Edit graphic... button in a graphic part's properties dialog or direct edit the graphic part to invoke the Bit Editor.

The Bit Editor can edit the existing bitmap or a bitmap from a file. If you have your own bitmaps stored in BMP files, you can bring them in by using the Open... command in the Bit Editor's File menu.

Apply Button

When you press the Apply button, the bitmap is updated and the part associated with the bitmap is also updated. Note that the File/Save command applies the changes to both the object and a file.

Reset Button

The Reset button throws out all changes that you have made since the last save or apply operation.

File Menu

- New: Clears the contents of the Bit Editor.
- Open...: Allows you to load a bitmap from a BMP file in Windows or OS/2 bitmap format.
- Save: Saves the bitmap in Windows or OS/2 BMP format.
- Save as...: Allows you to rename a file and specify a new path before saving.
- Exit: Closes the Bit Editor. Prompts you to save changes.
Appendix C: Standard Part Properties

Edit Menu

Select  Allows you to designate an area of the bitmap to cut or copy. Choose this command and then drag to define the area you want to select.
Cut     Cuts the area designated by the Select command to the clipboard.
Copy    Copies the area designated by the Select command to the clipboard.
Paste   Places a draggable rectangle in the center of the bitmap. To place it, drag the rectangle to the desired location. The clipboard contents will be placed at the destination rectangle.
Clear   Changes the entire graphic to the transparent screen color.
Fill    Changes all pixels of the same color that are connected to the one you click on, to the current drawing color.

Options Menu

Grid    Toggles grid display. The grid has vertical and horizontal lines which makes it easier to draw in the Bit Editor.
Change size… Use to change the size of the bitmap. A dialog pops up to prompt for new bitmap size. The bitmap will stretch if the new size is larger and shrink if the new size is smaller.
Change color… Lets you choose the current drawing color.

Edit icon Property

The Icon Editor allows you to edit icons of nonvisual parts, window parts, and the contents of static graphics. The Icon Editor displays the icon at normal size on the lower left and a magnified image of it on the right.

As you edit the individual magnified pixels on the right, the icon changes to reflect your edits. The following illustration shows the Icon Editor in Windows 95, which was invoked from the Properties dialog of the speaker part:
The following illustration shows the Icon Editor in OS/2, which was invoked from the Properties dialog of the application part:

![Icon Editor](image)

**Apply Button**
When you press this button, the icon is updated and the part associated with the icon is also updated. Note that the File/Save command applies the changes to both the object and a file.

**Reset Button**
When you press this button all changes that you have made since the last save or apply operation are discarded.

**Mode**
These radio buttons let you choose between the following three drawing modes:

- **Color**
  When you click on a pixel, it changes to the color last selected from the Color pull-down menu.

- **Screen**
  When you click on a pixel, it is marked as transparent. The icon editor indicates pixel transparency by setting the pixel to a certain color.

  You can view this color by selecting the Screen mode and looking at the Pen/Fill color. To avoid confusion, it is best not to select this color for drawing in Color mode.

- **Inverse**
  Draws with a different color to indicate that this area is inverted when the icon is moved or dropped.
Appendix C: Standard Part Properties

File Menu
New
Clears the contents of the Icon Editor.
Open...
Allows you to load an icon from a file in Windows or OS/2 icon format.
Save
Saves the icon in the operating system’s icon format.
Save as...
Allows you to rename a file and specify a new path before saving.
Exit
Closes the Icon Editor. Prompts you to save changes.

Edit Menu
The Edit menu does not support Cut, Copy, and Paste due to the absence of a publicly documented clipboard format for icons which would enable clipboard exchange between various icon editors.
Clear
Changes the entire icon to the transparent screen color.
Fill
Changes all pixels of the same color, that are connected to the one you click on, to the current drawing color.

Options Menu
Grid
Toggles grid display. The grid has vertical and horizontal lines which makes it easier to draw in the Icon Editor.
Test
Previews the icon by changing the cursor to display the current icon. To end the test mode and return the cursor to its normal shape, click anywhere in the Icon Editor.
Change color...
Lets you choose the current drawing color.
Appendix C: Standard Part Properties

Font Property

The Font dialog allows you to select the font used by visual parts for labels and/or text. The dialog is invoked by pressing the Font... button in a visual part's properties dialog.

In Windows, the following standard Font dialog appears:

To select a new font in Windows:
1. Double-click on a visual part that displays text, to open its properties dialog.
2. Click on the Fonts... button.
   
   A Font dialog is displayed.
3. Enter the font name in the Font: combo box or select one from the list.
4. Enter a style in the Font Style: combo box or select one from the list.
5. Enter a font size in the Size: combo box or select one from the list.
6. Click on the OK button when you are done.
7. In the Properties dialog, click on the OK button.

A sample of the currently selected font settings are displayed in the Sample box. The area directly below this tells you the type of font selected.

See your Windows documentation for more information on using the Font dialog.
In OS/2, the following Font dialog appears:

```
Font

Name
Courier

Style
Normal

Sample
AaBbYyZz

Size
10

Display

Printed

Emphasis

Outline

Underline

Strikeout
```

To select a new font in OS/2:

1. Double-click on a visual part that displays text, to open its properties dialog.
2. Click on the Fonts... button.
   
   A Font dialog is displayed.

3. Choose a font name in the font Name list.
4. Choose a style from the Style list.
5. Enter a font size in the Size: combo box or select one from the list.
6. Check one of the check boxes in the emphasis group pane, if desired, for underlining or strikethrough.
7. Click on the OK button when you are done.
8. In the Properties dialog, click on the OK button.

A sample of the currently selected font settings are displayed in the Sample box. The area directly below this tells you the type of font selected.

You can set the font for an entire group of parts at one time. See Setting Properties for a Group of Parts in chapter 4 of the Visual Smalltalk Enterprise Workbench User's Guide for information on this procedure.

**Forecolor Property**

The Forecolor property controls the foreground or text color of a visual part. Some visual parts, such as push buttons, don’t have color properties.

Pressing the Forecolor... button in a visual part's properties dialog invokes a standard color dialog. For further details, see the Backcolor Property topic earlier in this appendix.
Appendix C: Standard Part Properties

Order children Property

Graphical user interfaces (GUIs) use the concept of input focus. Input focus determines which part receives keyboard input. When you press a key on your keyboard, the response depends on which screen object has input focus at the time you press the key. If an object like a push button or a check box has input focus, it responds to keyboard input as though it has been clicked on with the mouse. An entry field, on the other hand, generates an entered event when you press the Enter or Return key.

One way to change the input focus is by clicking on a new object with the mouse. A much faster way to change the input focus when navigating complicated forms is to use the Tab key. PARTS Workbench automatically assigns a specific order, called the tab order, to the child parts of each window, dialog window, and notebook.

You can also organize parts into a group which is treated as a single stop in the tab order. Note that you cannot set tab stops for radio buttons.

Each time you press the Tab key at runtime, the input focus shifts to the next object or group of objects in the specified order. If you press the Tab key enough times, you will return to the original object. You change the input focus within a group using the Arrow keys.

You can modify the tab order and define groups using the Tab and Direction key property of windows, dialog windows, and notebooks. You can access this dialog by clicking on the Order Children... button in the Properties dialog of windows, dialog windows, notebooks, and application parts.

To modify the tab order of a parent part's children, perform the following steps:

1. Double-click on the parent part.
   A Properties dialog is displayed.

2. Click on the Order Children... button.
   A Tab and Direction Keys dialog is displayed.

3. Select a part name from the list pane and drag it to a new location in the list.

4. Click on the OK button.

5. In the Properties dialog, click on the OK button.
To remove a child part from the tab order, perform the following steps:

1. Double-click on a parent part.
   A Properties dialog is displayed.

2. Click on the Order Children... button.
   A Tab and Direction Keys dialog is displayed.

3. Select a part name from the list pane.
   Parts that are currently included in the tab order have an asterisk preceding their name.

4. Click on the Clear Tab Stop button.
   The asterisk preceding the part’s name disappears.

5. Click on the OK button.

6. In the Properties dialog, click on the OK button.

To add a child part to the tab order, perform the following steps:

1. Double-click on a parent part.
   A Properties dialog is displayed.

2. Click on the Order Children... button.
   A Tab and Direction Keys dialog is displayed.

3. Select a part name from the list pane.
   If a part does not have an asterisk preceding its name, it is not currently part of the tab order. Note that all children are initially included in the tab order.

4. Click on the Set Tab Stop button.
   An asterisk is displayed preceding the part’s name.

5. Click on the OK button.

6. In the Properties dialog, Click OK.

To create a group of parts, perform the following steps:

1. Double-click on a parent part.
   A Properties dialog is displayed.

2. Click on the Order Children... button.
   A Tab and Direction Keys dialog is displayed.

3. Select a part name.
   By default, there is always at least one, undeletable group that begins with the first part in the tab order list pane. If you select the first part, you will notice that both the Start Group and Clear Group buttons are disabled.

4. Click on the Start Group button.
   A line appears above the selected part. All the parts from the selected part to the next line or the end of the list, whichever comes first, are part of the same group.
When you tab to the group, the input focus goes to the first item in the group. Pressing the Tab key again advances the input focus to the next part outside the current group or to the next group. If you want to move the input focus to another part in the same group, use the direction (Arrow) keys.

5 Click on the OK button.
6 In the Properties dialog, click the OK button.

**To clear a group, perform the following steps:**

1 Double-click on a parent part.
   
   A Properties dialog is displayed.
2 Click on the Order Children... button.
   
   A Tab and Direction Keys dialog is displayed.
3 Select a part name appearing directly below a horizontal line in the list pane.
4 Click on the Clear Group button.
   
   The horizontal line disappears.
5 Click on the OK button.
6 In the Properties dialog, click the OK button.

**Setting Tab Order with Events and Messages**

You can also set tab order with events and messages. Most visual parts have **tabbed** events that are triggered when the part has the input focus and the Tab key is pressed. You can program the tab order for the parts in your application by creating links between the **tabbed** event of the first part and the **setFocus** message of the second part, and so on.

If a part has a link fired by the part’s **tabbed** event, tab stops set for that part using the Tab and Direction Key dialog are ignored.
Part name Property

Part names are used for identification purposes. They appear as the icon label of most nonvisual parts, in the Workbench’s information area when you place the cursor over the part, in the Select Parts dialog’s list of parts, and when printing a textual description of an application.


All parts in the same application must have a unique part name that begins with a capital letter. The rest of the characters must be letters or numbers. The first letter of the name must be capitalized. By convention, each embedded word in the name is also capitalized (for example, PartName).

A part name is automatically assigned to a part when you add it to an application, but you can change it at any time. Default names consist of the generic part name and its order of creation. For example, if an application has three list panes, they are named ListPane1, ListPane2, and ListPane3 by default.

To change a part’s default name using the properties dialog, perform the following steps:

1. Double-click on the part.
   The part’s Properties dialog is displayed.

2. Enter a new name in the Part name: entry field and click the OK button.
   If you attempt to assign an invalid part name, PARTS Workbench will open a warning dialog suggesting an alternative name. You may either accept the suggestion or cancel the dialog and assign a different name.

   or

1. Select the part.
2. Choose Selected/Edit properties from the Workbench menu bar.

3. Enter a new name in the Part name: entry field and click the OK button.
   If you attempt to assign an invalid part name, PARTS Workbench will open a warning dialog suggesting an alternative name. You may either accept the suggestion or cancel the dialog and assign a different name.

You can also change the name of some parts via direct-editing.
Size window Property

The Window layout property dialog, accessed through the Size window... button in a visual part's Properties dialog, lets you specify the layout attributes of a part. The layout attributes determine a child part's behavior when the size of its parent (containing part) is changed. These attributes are platform independent; the same attribute values would work on OS/2, Windows, or some other platform.

The Window Layout dialog has two main sections. The top section is a set of radio buttons that allow you to choose one of the most often used predefined layouts, or customize the settings. The bottom section enables you to define the customized settings.

Location fixed, size fixed

This is the default layout for all parts in the PARTS catalog. When you resize any parent of a part with this layout, the child part's top left corner will maintain the same distance from the top left corner of its containing (parent) part. And the size (width and height) of the part never changes.

All sides proportional to parent

When you resize any parent of a part with this layout, all of the child's four sides keep the same proportion relative to its parent's size. In other words, the four sides stretch in proportion as you stretch its parent.

All sides fixed inset from parent

When you resize any parent of a part with this layout, the child part's four sides will remain the same distance from its parent's four sides. In other words, the four sides will stretch as you stretch its parent, but the distances between the four sides and its parent's four sides are fixed rather than proportional.
Appendix C: Standard Part Properties

Customized
Selecting the Customized radio button allows you to define the parts behavior in the lower section of the dialog.

Units of Measurement
The Measurement unit combo box, located below and to the right of the Customized radio button, lets you select the units of measurement to use for customizing Window Layout attributes. There are four units of measure supported: twip, inch, centimeter, and pixel. Twip is the default.

A twip is equal to 1/1440 of a logical inch which is a real inch on the printer but an approximation on the screen (based on its font resolution). Since inches and centimeters are rather large measurements, you ordinarily specify fractions when you use them. Twip, inch, and centimeter are device and resolution independent and therefore they are preferred units to use.

A pixel is the smallest display element on the screen. When you use pixel, your application will look different than you intended on screens with different resolutions than the one you used to develop your application. It is therefore not recommend that you use pixel as the unit of measurement under most circumstances.

Avoid using System font when using twip, inch, and centimeter because you have no control over the point size on a screen with different resolution than the one you used to develop the application.

Attributes of Customized Settings
The lower section of the Window Layout dialog has two group panes (horizontal attributes and vertical attributes) showing all possible combinations of the attributes (by radio buttons) as well as their values (in the entry fields).

Although each dimension has four attributes, you can only select exactly two (no more, no less) to represent a part's layout. The number of attributes (two) and the valid combinations are enforced by the radio buttons.

When an attribute is selected, its current setting is displayed in the entry field directly above its radio button. If the Fixed radio button is selected, the value is a number expressed in the currently selected measurement units.

If the Proportional button is selected, the value is a fraction of its fixed value over the width of the parent part. When the value is a Proportional, it is displayed in blue; otherwise it is black.

Once you have selected a pair of attributes, you can change the values in the entry fields for the selected attributes. The number that you type in can be any valid number: an integer, a floating point number, or a fraction.

After you have changed any value and clicked on the OK button for both the Window Layout dialog and the Property dialog, the selected part is repositioned and/or resized according to the new values.
Appendix C: Standard Part Properties

Horizontal Attributes
The horizontal attributes are:
• left inset—the distance between a part’s left side and its parent’s left side
• width—the distance between a part’s left side and its own right side
• right inset—the distance between a part’s right side and its parent’s right side
• horizontal center—the distance between a part’s center and its parent’s left side
The valid combinations of horizontal attributes are:
• left inset and width
• width and right inset
• left inset and right inset
• width and horizontal center

Vertical Attributes
The vertical attributes are:
• top inset—the distance between a part’s top side and its parent’s top side
• height—the distance between a part’s top side and its own bottom side
• bottom inset—the distance between a part’s bottom side and its parent’s bottom side
• vertical center—the distance between a part’s center and its parent’s top side
The valid combinations of vertical attributes are:
• top inset and height
• height and bottom inset
• top inset and bottom inset
• height and vertical center

Fixed or Proportional Attributes
You can make an attribute either fixed or proportional by selecting the Fixed or Proportional radio button located under the attribute’s name and entry field. If the attribute is fixed, then its value is a measurement between two sides. If the attribute is proportional, then its value is a ratio of its fixed value over its containing (parent) part’s width or height.

For example, a fixed left inset specifies the distance between the left sides of the child part and the parent part, whereas a proportional left inset indicates the ratio between the child part’s fixed inset over the parent part’s width.
Symbols
- (dash) 375
  - (minus sign) message 340
#pragma pack directive 72
& (ampersand)
  check box shortcut keys 89
  menu label shortcut keys 305
  notebook page shortcut keys 337
  push button shortcut keys 422
  radio button shortcut keys 427
  static text shortcut keys 465
  tab control page (Win32 only) shortcut keys 498
( ) (parentheses)
  parameter markers 482
* (asterisk). See also Password property 185
+ message 340
... (ellipsis)
  number holders 339, 349
  string holders 473, 478
/ (slash) 145
/ message 340
// message 340
: (colon)
  DLL accessor message names 187, 189
  following drive letters 181
  poke-item/hot-link item names 154, 160
< (less than) message
  number holders 341
  string holders 474
< > angle brackets 482
<= (less than or equal) message
  number holders 341
  string holders 474
> (greater than) message
  number holders 341
  string holders 474
>= (greater than or equal) message
  number holders 341
  string holders 474
@ message 341
{ } square brackets 481, 482
\ message 341
(underscores) 375

A
Abort button 113
Abort/Retry/Ignore confirmation dialogs 113
abortChange message
  alphabetic entry fields 26
  button list box 65
  currency entry field 140
  date entry field 147
  drop-down list 200
  entry field 212
  float entry field 230
  integer entry field 260
  list panes 276
  multiple choice lists 312
  notebooks 329
  phone number entry field 370
  picture entry field 377
  tab control (Win32 only) 487
  value set (OS/2 only) 587
abortClose message
  aboutToClose event and 168
dialog window 170
windows 606
aboutToChange event
  button list box 63
drop-down list 200
list panes 276
multiple choice lists 312
notebooks 328
tab control (Win32 only) 486
value set (OS/2 only) 586
aboutToChangeLabel:to: event
  list view (Win32 only) 285
tree view (Win32 only) 570
aboutToChangeTo: event
  alphabetic entry field 25
Index

currency entry field 139
date entry field 146
entry field 211
float entry field 229
integer entry field 259
phone number entry field 369
picture entry field 376
aboutToClose event
  abortClose message and 168
dialog window 170
windows 605
aboutToDelete: event
  container (OS/2 only) 118
aboutToEditLabel: event
  list view (Win32 only) 285
tree view (Win32 only) 570
aboutToUnload event
  part accessor 366
abs message 341
abs: message
  computation 105
absolute values 105, 341
Accelerator property
  menu text item 309
Access by property
  table pane 512
accessing
  drop-down list items 199
  notebook pages 327
  part properties 631
  Properties dialogs xxiv
  table pane data 501, 512
  Window Layout property dialog 644
Add... button
  DLL accessor 189
add: message
  currency entry field 140
  float entry field 230
  integer entry fields 260
  ordered collection holders 355
add:after: message
  ordered collection holders 355
add:afterIndex: message
  ordered collection holders 356
add:before: message
  ordered collection holders 356
add:beforeIndex: message
  ordered collection holders 356
addAll: message
  ordered collection holders 356
addDragFormat: message
  container (OS/2 only) 120
addFirst: message
  ordered collection holders 356
adding
  See also addition; creating; defining
+ message 340
check marks to menu text items 307
child parts to tab order 641
combo box selection list items 97
drop-down list items 201
edit menus to text panes 525
fields to Btrieve record definitions 55
hot-link item events 153
hot-link item messages 160
key/value pairs to dictionaries 179, 180
list pane items 277, 278
messages to DLL accessors 189
multiple choice list items 311, 313
numbers to
  currency entry field values 140
  float entry field values 230
  integer entry field values 260
  number holder values 340, 342
  ordered collection items 360
  pages to notebooks 327
  poke-item events 159
  poke-item messages 154
  sum:and: message
  computation 106
  values to ordered collection holders 355
addLast: message
  ordered collection holders 356
addMechanism: format: message
  container (OS/2 only) 120
Adjust column widths property
  table pane 512
Adjust row heights property
  table pane 512
alarm message
  speaker 449
Alignment property
  container (OS/2 only) 132
  integer entry field 265
  static text 464
Alignment property (OS/2 only)
  alphabetic entry fields 29
  currency entry field 143
  entry field 214
  float entry field 233
Alignment property (OS/2)
  phone number entry field 372
alignTitleCenter message
  container (OS/2 only) 120
alignTitleLeft message
  container (OS/2 only) 120
alignTitleRight message
  container (OS/2 only) 120
All sides fixed inset from parent radio button of Size window property 644
All sides proportional to parent radio button of Size window property 644
alphabetic entry fields
  See also currency entry fields; date entry fields; entry fields; float entry fields;
  integer entry fields; phone number entry fields; picture entry fields
Btrieve field and data type 50
Always Show Selection property
  list view (Win32 only) 289
ampersand (&)
  check box shortcut keys 89
  menu label shortcut keys 305
notebook page shortcut keys 337
push button shortcut keys 422
radio button shortcut keys 427
static text shortcut keys 465
  tab control page (Win32 only) shortcut keys 498
angle brackets < > 482
append: message
  rich edit (Win32 only) 433
text pane 520
appendToValue: message
string holders 474
Application file name property
  property interface 406
Application property
  DDE client 153
  DDE server 159
application reuse modes 320
applications
  See also application reuse modes; nested parts; part accessors; reusing clipboard operations 91
  DDE client/server 151, 155, 157, 159
  improving runtime performance 364
  splitting functionality of 365
Apply button (Bit Editor) 634
Apply button (Icon Editor) 636
apply: event
  tab control page (Win32 only) 495
arcCos message 341
arcSin message 341
arcTan message 342
Argument type property
  DLL accessor 191
Arguments property
  DLL accessor 190
arrangeItems message
  container (OS/2 only) 120
array constants 41
array holders
  See also dictionary holders; number holders; ordered collection holders;
  string holders; value holders
  converting ordered collection holder values to 356
  placing Dictionary values in 176
arrays, creating 135
asArray message
  array holder 38
  dictionary holder 176
  ordered collection holders 356
  string holders 474
asArray: message
  container (OS/2 only) 129
  conversion 135
asArrayOfSubstrings message
  string holders 474
asAsciiZ message
  string holders 475
asCharacter message 342
ASCII (American Standard for Information Interchange) values, converting number holder values to 342
asDate message
  string holders 475
asDate: message
  conversion 135
asFloat message
  number holders 342
  string holders 475
asFloat: message
  conversion 136
asInteger message
  string holders 475
asInteger: message
  conversion 136
asLowerCase message
  string holders 475
asLowerCase: message
  conversion 136
asNumber: message
  conversion 136
asOrderedCollection message
  array holder 38
  dictionary holder 176
  string holders 475
asParameter message
  string holders 476
asSet message
  array holder 38
  dictionary holder 176
  ordered collection holders 356
  string holders 476
Index

asSortedCollection message
array holder 38
dictionary holder 176
ordered collection holders 357
string holders 476
asString message
string holders 476
asString: message
conversion 136
asSymbol message
string holders 476
asterisk (*). See also Password property 185
asUpperCase message
string holders 476
asUpperCase: message
conversion 136
@ message 341
at: aKey put: message
dictionary holder 176
at: message
array holder 38
class Dictionary 175
dictionary holder 176
ordered collection holders 357
string holders 476
at:put: message
array holder 39
class Dictionary 175
ordered collection holders 357
string holders 476
audio message
AVI video (Win32 only) 43
audio: message
AVI video (Win32 only) 43
CD audio (Win32 only) 79
audioLeft: message
CD audio (Win32 only) 79
audioRight: message
CD audio (Win32 only) 79
Auto Arrange property
list view (Win32 only) 289
Auto check box property
check box 88
Auto Save On Close property
toolbar (Win32 only) 540
Auto scroll property
currency entry field 143
date entry field 149
entry field 214
float entry field 233
integer entry field 262
phone number entry field 372
picture entry field 379
automatic check boxes 85, 88
automatic file loading 364
Automatic radio button property
radio button 426
automatic radio buttons 423, 426
Automatically arrange items property
toolbar (OS/2 only) 151
autoSaveOnClose: message
toolbar (Win32 only) 536
B
Back pages property (OS/2)
notebooks 332
back: event
tab control page (Win32 only) 495
Backcolor standard property 632
beginAdjust event
toolbar (Win32 only) 535
beginJob message
printer 381
between: min and: message 342
Binder type property (OS/2)
notebooks 332
Bit Editor
graph panes 241
static graphics 461
Bitmap Height property
toolbar (Win32 only) 539
bitmap message
drawn button 196
graph pane 240
menu bitmap item 298
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
Bitmap property (OS/2 only)
static graphic 462
bitmaps
See also graph panes; menu bitmap items;
static graphics
clipboard operations 91
notebook page tabs 337
printing 381
returning drawn button 196
setting graph panes to contents of 241
bitsPerSample message
wave audio (Win32 only) 597
blanks
converting control characters to 136
removing
from prompts 392, 393
from string holder contents 478
replacing control characters with 477
BMP files
opening 634
.bmp files. See bitmaps
boolean values
  check boxes 85, 87
  comparison messages 101
Border property
  See also Dialog border property
  alphabetic entry field 28
  currency entry field 143
  date entry field 149
  dial pane 163
  dialog window 172
  entry field 214
  float entry field 233
  graph pane 241
  integer entry field 262
  OLE control (Win32 only) 354
  phone number entry field 372
  picture entry field 379
  rich edit (Win32 only) 438
  text pane 523
  value set (OS/2 only) 589
  window 608
Border property (OS/2 only)
  push button 421
Browse property
  property interface 406
  Btmap Width property
  toolbar (Win32 only) 539
  BTREXmpl.PAR file 62
Button labels property
  confirmer 111
  confirmer with cancel 116
Buttons property
  header control (Win32 only) 247
  tree view (Win32 only) 574
bytesPerSecond message
  wave audio (Win32 only) 597
C
C structures
  See also DLL Accessor
  DLL calls and 192
callException:arguments: event
  DLL accessor 188
Calling convention property
  DLL accessor 190
Cancel button
  confirmation with cancel dialogs 113
  triggering no event 109
cancel event
  confirmer with cancel 114
  file dialog 223
  prompter 391
ceiling message. See also floor message 342
Cell height property
  table pane 511
  cell index numbers 501
Cell width property
  table pane 511
  cellAtName: message
  table pane 506
  cellAtPoint: message
  table pane 506
  cellAtRow:column: message
  table pane 506
  cellAtRow:column:put: message
  table pane 506
  cellChanged: event
  table pane 503
  cellChangedAtRow:column:value: event
  table pane 503
  cellEnteredAtRow:column:value: event
  table pane 503
  cells, table pane
    accessing 501
    direct-editing 510
    editing 503
    selecting 502
Change color (Icon Editor Option menu
cmd) 637
Change color... (Bit Editor Options menu)
cmd 635
Change file... property
  nested part 325
Change OLE control property
  OLE control (Win32 only) 353
Change size... (Bit Editor Options menu)
cmd 635
changeColor message
  rich edit (Win32 only) 433
changed Coordinate: event
  value set (OS/2 only) 586
changed event
  property interface 404
  toolbar (Win32 only) 535
changed: event
  alphabetic entry fields 25, 30
  button list box 64
  combo box 96
  container (OS/2 only) 118
  currency entry field 140
  date entry field 146
  dial pane 162
  drop-down list 200
  entry field 211
  float entry field 230
  graph pane 239
  integer entry fields 259
Index

list panes 276
list view (Win32 only) 285
multiple choice lists 312
phone number entry fields 370
picture entry fields 376
progress bar 385
rich edit (Win 32 only) 432
scroll bars (Win32 only) 439
sliders (OS/2 only) 444
spin buttons (OS/2 only) 453
string holders 473
tab control (Win32 only) 486
table pane 504
text pane 520
track bar 561
tree view (Win32 only) 570

changedElement: event
array holder 37
changedIndex: event
button list box 64
drop-down list 200
list panes 276
value set (OS/2 only) 587
changedkey event
automatic generation of 180
changedLabel:to: event
list view (Win32 only) 285
tree view (Win32 only) 570
changedPage: event
notebooks 328
changedPageNumber: event
notebooks 329
tab control (Win32 only) 487
changeFont message
rich edit (Win32 only) 433
changeIconForObjects: toIcon: message
container (OS/2 only) 120
changeIconForPaths: toIcon: message
container (OS/2 only) 120
changeNameFor: toName: message
container (OS/2 only) 121
changeParagraph message
rich edit (Win32 only) 433
changeTabs message
rich edit (Win32 only) 433
changeValueBy: message 342
changing
See also changed: event; direct-editing;
editing; property editing
application reuse modes 320
application window size 31

array holder icon names 40
Btrieve field definitions 57
check box labels 87
dial pane value 161
dictionary holder values 180
drop-down list items
  selections of 199
setting new 201
float entry field values 232
individual tick marks of dial pane 165
input focus 640
list pane item selections 275
menu text item labels 309
notebook page binder type 332
notebook page tab shapes 332
number holder values 339
size of table pane columns/rows 503
slider values 444
static graphics at runtime 461
value set (OS/2 only) cell selections 586
changing: event
track bar 561
channels message
wave audio (Win32 only) 597
characters
See also Maximum number of characters
  property; string holders
case of entry field 214
converting
to lowercase 136, 475
  number holder values to 342
to uppercase 136, 476
defining, for picture entry fields 379
deleting individual, from entry fields 210
check boxes
See also group panes; radio buttons
Btrieve field and data type 50
check marks
displaying with menu text items 308
  menu bitmap items 297
  menu text items 307
check message
check box 86
check boxes 86
  menu bitmap item 298
  menu text item 308
triggering checked event 86
checked event
check box 86
checked: event
button list box 64
checkedIndex: event
button list box 64

652

Visual Smalltalk Enterprise Part Reference
checked/isTrue event
menu bitmap item 297
menu text item 307
checkIndex: message
button list box 65
checkItem: message
button list box 65
child parts
adding to tab order 641
behavior when parent size is changed 644
ordering tab order of nested 640
removing from tab order 641
CHKBXMPL.PAR file 90
Class property
value holder 580
classes
conversion part and 135
setting value holder 578
typed value holders and 577
CLBRXMPL.PAR file 93
Clear (Bit Editor Edit menu) command 635
Clear (Icon Editor Edit menu) command 637
clear message
alphabetic entry fields 26
combo box 96
currency entry field 140
date entry field 147
entry field 212
float entry field 231
integer entry fields 260
phone number entry fields 370
picture entry fields 377
rich edit (Win32 only) 433
table pane 506
text pane 520
Clear modified ticks property
sliders (OS/2 only) 448
Clear tick labels property
sliders (OS/2 only) 448
clearBitmap message
graph pane 240
clearIndeterminate message
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
clearModified message
rich edit (Win32 only) 433
text pane 521
clearSelection message
rich edit (Win32 only) 433
text pane 521
click message
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
clicked event
drawn button 195
menu bitmap item 297
menu label 303
menu text item 307
push buttons 419
toolbar button (Win32 only) 544
clickedIndex: event
clearBitmap message
graph pane 240
header control (Win32 only) 245
list view (Win32 only) 285
radio buttons 424
toolbar (Win32 only) 535
toolbar toggle button (Win32 only) 554
collapsedIndex: event
header control (Win32 only) 245
list view (Win32 only) 285
clipboard accessors
See also text panes
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
close message
AVI video (Win32 only) 44
Btrieve accessor 51
CD audio (Win32 only) 80
dialog window 170
DLL accessor 188
DLL files 187
wave audio (Win32 only) 598
windows 606
closed event
dialog window 170
information dialogs 255
windows 606
closeDoor message
CD audio (Win32 only) 80
closing
Btrieve database files 51
DLL files 187, 188
CMBOXMPL.PAR file 100
collapsed: event
container (OS/2 only) 118
collapsedItem: message
tree view (Win32 only) 571
collapsedTreeForItem: message
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
collections
concatenating 39
converting string holders to 476
returning sorted 137
table panes 501

Visual Smalltalk Enterprise Part Reference
Index

colon (:)
   DLL accessor message names 187, 189
   following drive letters 181
   poke-item/hot-link item names 154, 160
Color drawing mode of Icon Editor 636
Column label height property
table pane 511
columnAt: message
table pane 506
columnAt:put: message
table pane 506
columnClicked: event
   list view (Win32 only) 285
Columns property
   value set (OS/2 only) 589
Columns to left of split bar property
   container (OS/2 only) 133
columns, table pane, adjusting width of 512
columnSelected: event
table pane 504
columnWidths message
   header control (Win32 only) 246
combo boxes
   See also drop-down lists; entry fields; list pans
Command property
   launch pad 267
COMPARE.PAR file 104
COMPUTE.PAR file 107
concatenate: message
   appendToValue message vs. 474
   array holders 39
   ordered collection holders 357
   string holders 477
concatenating
   collections 39
   ordered collections 357
conditional messages 270
   link junction part 273
configure message
   AVI video (Win32 only) 44
CONFIRM.PAR file 112
confirmation dialogs. See confirm part;
   confirm with cancel part
confirmation with cancel dialogs. See confirm part
   with cancel parts
CONFIRMCP.PAR file 116
confirm parts
   See also confirm with cancel parts
confirm with cancel parts
   See also confirm parts
described 113
constants. See value holders

Content properties...
   property
   nested part 325
contents message
   header control (Win32 only) 246
   list view (Win32 only) 287
   tree view (Win32 only) 571
   video pane (Win32 only) 593
Contents property
   alphabetic entry fields 28
   currency entry field 142
   date entry field 148
   entry field 213
   float entry field 232
   integer entry field 262
   phone number entry field 372
   picture entry field 379
   status window (Win32 only) 470
contents: message
   header control (Win32 only) 246
   list view (Win32 only) 287
   tree view (Win32 only) 571
   video pane (Win32 only) 593
context-sensitive help 250
control characters 136
Control menu property (Windows)
   window 608
Control properties property
   OLE control (Win32 only) 354
Control-menu box property (Windows)
   dialog window 172
controlTabbed event
   rich edit (Win32 only) 432
   text pane 520
conventions, manual xxi
conversion part 135
Convert characters to property (Windows only)
   entry field 214
CONVERT.PAR file 138
converting
   See also conversion parts
   array holder contents to collection class
   members 38
   Btrieve status codes to text 51
   characters
      to lowercase 136, 475
      to uppercase 476, 136, 476
   collections to ordered collections 355
   control characters to blanks 136
   data types 135
   degrees to radians 343
   Dictionary values to other collections 176
   number holder values to characters 342
   ordered collection holder values to other
   collections 356

Visual Smalltalk Enterprise Part Reference
radians to degrees 347
string holder contents to other data types 475
coordinates, screen
computing relative positions of 339
number holders and 341
table pane cells 505, 506
x values, messages returning 345, 348
y values, messages returning 343, 344, 346, 349
Copy (Bit Editor Edit menu) command 635
copyTo: message
disk accessor 218
copyBitmap: message
clipboard accessor 91
copyBitmapFromScreen message
clipboard accessor 92
copyException event
clipboard accessor 91
copy messages triggering 92
copyFrom: message
ordered collection holders 357
copyFromTo: message
array holder 39
string holders 477
copying
See also clipboard accessor
bitmaps to clipboard 91
entry field text to clipboard 210
failures 91
files 218
graph pane contents to clipboard 240
text from rich edit (Win32 only) to clipboard 433
text from text panes to clipboard 521
copyReplaceFromTo:with: message 39
ordered collection holders 357
string holders 477
copySelection message
rich edit (Win32 only) 433
text pane 521
copyString: message
clipboard accessor 92
copyToClipboard message
graph pane 240
cos message 342
Create Field dialog
C structure 75
Create field... property
C structure 75
Create form parts... property
Btrieve accessor 57
C structure 76
create: message
disk accessor 182
creating
applications 32
arrays 39, 135
Btrieve files 50
Btrieve record definitions 55
check box labels 88
DDE server applications 157
dialog window requiring answer 168
dialog window titles 171
directories 182
groups of parts 641
ordered collections 357, 358
sorted collections 137
submenus 307
substrings 474
window titles 607, 608
curly braces { } 481, 482
currency entry fields
See also alphabetic entry fields; date entry fields; entry fields; float entry fields; integer entry fields; phone number entry fields; picture entry fields
Btrieve field and data type 50
currentDate message
timer 527
currentRowBottom message
table pane 506
currentRowDown message
table pane 507
currentRowTop message
table pane 507
currentRowUp message
table pane 507
currentTime message
timer 527
currentTrack message
CD audio (Win32 only) 80
currentTrack: message
CD audio (Win32 only) 80
cursor, mouse, entry fields and 209
cursoredObject message
container (OS/2 only) 121
Customizable property
toolbar (Win32 only) 538
customize message
toolbar (Win32 only) 536
Customized radio button of Size window property 645
Cut (Bit Editor Edit menu) command 635
cutSelection message
rich edit (Win32 only) 434
text pane 521
cutting. See also clipboard accessors 521

dash (-) 375
data structures, packing DLL 72
data types
  C language 76
  converting between 135
data types, Btrieve field types vs. PARTS 50, 58
date entry fields
  See also alphabetic entry fields; currency entry fields; entry fields; float entry fields; integer entry fields; phone number entry fields; picture entry fields
  alphabetic entry fields and 29
  described 145
date: message
  file accessor 218
dates
  asDate message returning 135
  currentDate message returning 527
  slash (/) separating 145
dateTimeString: message
  file accessor 218
DDE Client part. See also DDE Server part 151
DDE conversations
  described 155
  initiating 154, 160
decrementPosition message
  progress bar 386
  track bar 562
decrementPositionBy: message
  progress bar 386
  track bar 562
decrementValue message 342
decompressInUse: message
  container (OS/2 only) 121
Default button property
  push button 421
Default button property (OS/2 only)
  drawn button 197
Default file name property
  Btrieve accessor 54
Default pattern property
  file dialog 225
Default property
  prompter 393
Default tick length property
  dial pane 164
  sliders (OS/2 only) 447
defaultMessage message
  prompter 391
defaultPrinterName message
  printer 381

defining
  Btrieve database records 49
  C structure fields 71, 75
  characters for picture entry fields 379
  check box shortcut keys 89, 305
  check box styles 88
  currency formatting 139
  DDE conversation topics 153, 159
  dictionary keys 179
  hot-link item events 151, 157
  hot-link item messages 157
  menu text item keyboard equivalents 310
  padded C structures 73
  picture entry field pictures 375
  poke-item events 157, 159
  poke-item messages 151, 157
  reuse modes 322
degreesToRadians message 343
Delete button
  DLL accessor 189
delete message
  Btrieve accessor 51
delete: event
  container (OS/2 only) 118
delete: message
  disk accessor 182
  file accessor 218
deleteAll message
  button list box 65
  combo box 96
  container (OS/2 only) 121
  drop-down list 200
  list panes 276
  multiple choice lists 312
  tree view (Win32 only) 572
deleteItem: message
  button list box 65
  combo box 96
  container (OS/2 only) 121
  drop-down list 200
  list panes 277
  multiple choice lists 312
  tree view (Win32 only) 572
deleteItems: message
  container (OS/2 only) 121
deleteKey message
  text pane 521
deleteSelection message
  container (OS/2 only) 121
deleting
  See also cutting; removing
  applications 32
  Btrieve database records 51
  combo box selection list items 96
dictionary holder contents 177
directories 182
DLL accessor messages 189
drop-down list items 200
entry field text 210
fields from Btrieve record definitions 56
files 218
hot-link item events 154
hot-link item messages 160
key/value pairs from dictionaries 180
list pane items 276
poke-item events 160
poke-item messages 154
rich edit (Win32 only) text 433
text pane text 521
delivering applications
DELIVERY.TXT file 32
EXE files 364
DELIVERY.TXT file
Btrieve accessor licensing 58
delivering applications 32
Demand load all pages property
tab control (Win32 only) 491
Demand load property
notebooks 352
denominator message. See also numerator
message 343
Description... property
application 34
descriptionString: message
file accessor 219
deselectAll message
container (OS/2 only) 121
multiple choice lists 312
deselected event
tab control (Win32 only) 496
deselectIndex: message
multiple choice lists 313
deselecting multiple choice list items 311, 312
deselectItem: message
container (OS/2 only) 121
multiple choice lists 313
deselectItems: message
container (OS/2 only) 122
detach message
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
Development Help file property
application 34
Development Help topic property
application 34
dial panes
See also scroll bars
Btrieve field and data type 50
Dialog border property
dialog window 172
dialog boxes. See confirmers; confirmers
with cancel parts; dialog windows; modal
dialogs
Dialog label property
dialog window 172
dialog windows
See also push buttons; windows
Default button property 421
default drawn buttons 197
described 167
DIALOGWI.PAR file 174
DIALXMO.PAR file (OS/2 only) 166
DIALXMPW.PAR file (Win32 only) 166
dictionary holders. See also array holders;
number holders; ordered collection holders;
string holders; value holders 175
Dictionary, class
converting values to other collections 176
messages 175
difference:and: message
computation 105
dimmed characters. See disable message
Direct Edit property
list view (Win32 only) 288
tree view (Win32 only) 573
direct-editing
alphabetic entry fields 27
array holder 40
AVI video (Win32 only) 47
Btrieve accessor 54
button list box 67
C structure 74
CD audio (Win32 only) 83
check box 87
check boxes 87
clipboard accessor 92
combo box 97
comparison 102
computation 106
confirmers 111
confirmers with cancel 115
container (OS/2 only) 129
to 137
conversion 137
currency entry field 142
date entry field 148
dDE client 153
dDE server 159
dial pane 162
dialog window 171
dictionary holder 178
disk accessor 185
dLL accessor 188
<table>
<thead>
<tr>
<th>drawn button</th>
<th>196</th>
</tr>
</thead>
<tbody>
<tr>
<td>drop-down list</td>
<td>202</td>
</tr>
<tr>
<td>entry field</td>
<td>213</td>
</tr>
<tr>
<td>file accessor</td>
<td>220</td>
</tr>
<tr>
<td>file dialog</td>
<td>224</td>
</tr>
<tr>
<td>file menu</td>
<td>227</td>
</tr>
<tr>
<td>float entry field</td>
<td>232</td>
</tr>
<tr>
<td>format menu</td>
<td>236</td>
</tr>
<tr>
<td>graph pane</td>
<td>241</td>
</tr>
<tr>
<td>group pane</td>
<td>243</td>
</tr>
<tr>
<td>header control (Win32 only)</td>
<td>247</td>
</tr>
<tr>
<td>help menu</td>
<td>252</td>
</tr>
<tr>
<td>information dialog</td>
<td>257</td>
</tr>
<tr>
<td>integer entry fields</td>
<td>262</td>
</tr>
<tr>
<td>launch pad</td>
<td>266</td>
</tr>
<tr>
<td>link junction part</td>
<td>272</td>
</tr>
<tr>
<td>list panes</td>
<td>278</td>
</tr>
<tr>
<td>list view (Win32 only)</td>
<td>288</td>
</tr>
<tr>
<td>menu bitmap item</td>
<td>299</td>
</tr>
<tr>
<td>menu item separator</td>
<td>302</td>
</tr>
<tr>
<td>menu label</td>
<td>304</td>
</tr>
<tr>
<td>menu text item</td>
<td>309</td>
</tr>
<tr>
<td>menus</td>
<td>325</td>
</tr>
<tr>
<td>multiple choice lists</td>
<td>314</td>
</tr>
<tr>
<td>nested parts</td>
<td>325</td>
</tr>
<tr>
<td>notebook pages</td>
<td>336</td>
</tr>
<tr>
<td>notebooks</td>
<td>331</td>
</tr>
<tr>
<td>number holders</td>
<td>349</td>
</tr>
<tr>
<td>OLE control (Win32 only)</td>
<td>352</td>
</tr>
<tr>
<td>ordered collection holder</td>
<td>359</td>
</tr>
<tr>
<td>overview</td>
<td>xxiii</td>
</tr>
<tr>
<td>part accessor</td>
<td>367</td>
</tr>
<tr>
<td>phone number entry fields</td>
<td>371</td>
</tr>
<tr>
<td>picture entry fields</td>
<td>378</td>
</tr>
<tr>
<td>printer parts</td>
<td>383</td>
</tr>
<tr>
<td>progress bar</td>
<td>387</td>
</tr>
<tr>
<td>prompter</td>
<td>393</td>
</tr>
<tr>
<td>property constructor</td>
<td>396</td>
</tr>
<tr>
<td>property interface</td>
<td>406</td>
</tr>
<tr>
<td>property sheet</td>
<td>412</td>
</tr>
<tr>
<td>push buttons</td>
<td>420</td>
</tr>
<tr>
<td>radio buttons</td>
<td>425</td>
</tr>
<tr>
<td>rich edit (Win32 only)</td>
<td>436</td>
</tr>
<tr>
<td>scroll bars (Win32 only)</td>
<td>440</td>
</tr>
<tr>
<td>sliders</td>
<td>445</td>
</tr>
<tr>
<td>speaker</td>
<td>450</td>
</tr>
<tr>
<td>spin buttons (OS/2 only)</td>
<td>455</td>
</tr>
<tr>
<td>static graphics</td>
<td>461</td>
</tr>
<tr>
<td>static text</td>
<td>464</td>
</tr>
<tr>
<td>status window (Win32 only)</td>
<td>468</td>
</tr>
<tr>
<td>string holders</td>
<td>478</td>
</tr>
<tr>
<td>string templates</td>
<td>482</td>
</tr>
<tr>
<td>tab control (Win32 only)</td>
<td>489</td>
</tr>
<tr>
<td>tab control page (Win32 only)</td>
<td>496</td>
</tr>
<tr>
<td>table pane</td>
<td>510</td>
</tr>
<tr>
<td>text panes</td>
<td>523</td>
</tr>
<tr>
<td>timer</td>
<td>528</td>
</tr>
<tr>
<td>tree view (Win32 only)</td>
<td>572</td>
</tr>
<tr>
<td>value holders</td>
<td>580</td>
</tr>
<tr>
<td>value set (OS/2 only)</td>
<td>588</td>
</tr>
<tr>
<td>wave audio (Win32 only)</td>
<td>601</td>
</tr>
<tr>
<td>windows</td>
<td>607</td>
</tr>
<tr>
<td>wizard (Win32 only)</td>
<td>612</td>
</tr>
<tr>
<td>directories. See Disk Accessor</td>
<td></td>
</tr>
<tr>
<td>directoryException: event</td>
<td>182</td>
</tr>
<tr>
<td>create: message triggering</td>
<td>182</td>
</tr>
<tr>
<td>exists: message triggering</td>
<td>183</td>
</tr>
<tr>
<td>directoryName: message</td>
<td>182</td>
</tr>
<tr>
<td>disk accessor</td>
<td>182</td>
</tr>
<tr>
<td>directoryPath: message</td>
<td>183</td>
</tr>
<tr>
<td>disable message</td>
<td></td>
</tr>
<tr>
<td>alphabetic entry fields</td>
<td>26</td>
</tr>
<tr>
<td>button list box</td>
<td>66</td>
</tr>
<tr>
<td>check box</td>
<td>86</td>
</tr>
<tr>
<td>combo box</td>
<td>96</td>
</tr>
<tr>
<td>container (OS/2 only)</td>
<td>122</td>
</tr>
<tr>
<td>currency entry field</td>
<td>141</td>
</tr>
<tr>
<td>date entry field</td>
<td>147</td>
</tr>
<tr>
<td>dial pane</td>
<td>162</td>
</tr>
<tr>
<td>drawn button</td>
<td>196</td>
</tr>
<tr>
<td>drop-down list</td>
<td>201</td>
</tr>
<tr>
<td>entry field</td>
<td>212</td>
</tr>
<tr>
<td>float entry field</td>
<td>231</td>
</tr>
<tr>
<td>graph pane</td>
<td>240</td>
</tr>
<tr>
<td>header control (Win32 only)</td>
<td>246</td>
</tr>
<tr>
<td>integer entry fields</td>
<td>260</td>
</tr>
<tr>
<td>list panes</td>
<td>277</td>
</tr>
<tr>
<td>list view (Win32 only)</td>
<td>287</td>
</tr>
<tr>
<td>menu bitmap item</td>
<td>298</td>
</tr>
<tr>
<td>menu label</td>
<td>304</td>
</tr>
<tr>
<td>menu text item</td>
<td>308</td>
</tr>
<tr>
<td>multiple choice lists</td>
<td>313</td>
</tr>
<tr>
<td>notebooks</td>
<td>329</td>
</tr>
<tr>
<td>OLE control (Win32 only)</td>
<td>352</td>
</tr>
<tr>
<td>ordered collection holder</td>
<td>359</td>
</tr>
<tr>
<td>overview</td>
<td>xxiii</td>
</tr>
<tr>
<td>part accessor</td>
<td>367</td>
</tr>
<tr>
<td>phone number entry fields</td>
<td>371</td>
</tr>
<tr>
<td>picture entry fields</td>
<td>378</td>
</tr>
<tr>
<td>printer parts</td>
<td>383</td>
</tr>
<tr>
<td>progress bar</td>
<td>387</td>
</tr>
<tr>
<td>prompter</td>
<td>393</td>
</tr>
<tr>
<td>property constructor</td>
<td>396</td>
</tr>
<tr>
<td>property interface</td>
<td>406</td>
</tr>
<tr>
<td>property sheet</td>
<td>412</td>
</tr>
<tr>
<td>push buttons</td>
<td>420</td>
</tr>
<tr>
<td>radio buttons</td>
<td>425</td>
</tr>
<tr>
<td>rich edit (Win32 only)</td>
<td>436</td>
</tr>
<tr>
<td>scroll bars (Win32 only)</td>
<td>440</td>
</tr>
<tr>
<td>sliders (OS/2 only)</td>
<td>444</td>
</tr>
<tr>
<td>spin buttons (OS/2 only)</td>
<td>454</td>
</tr>
<tr>
<td>static text</td>
<td>463</td>
</tr>
<tr>
<td>status window (Win32 only)</td>
<td>468</td>
</tr>
<tr>
<td>string holders</td>
<td>478</td>
</tr>
<tr>
<td>string templates</td>
<td>482</td>
</tr>
<tr>
<td>tab control (Win32 only)</td>
<td>489</td>
</tr>
<tr>
<td>tab control page (Win32 only)</td>
<td>496</td>
</tr>
<tr>
<td>table pane</td>
<td>507</td>
</tr>
<tr>
<td>text pane</td>
<td>521</td>
</tr>
</tbody>
</table>
toolbar (Win32 only) 536
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 554
track bar 562
tree view (Win32 only) 572
value set (OS/2 only) 587
video pane (Win32 only) 593
disabledIsTrue event
   menu bitmap item 297
   menu text item 307
disabling parts. See disable message
DIDSACC.PAR file 186
Display column labels property
table pane 512
Display row labels property
table pane 512
displaying
   check marks with
      menu items 297
      menu text items 308
column and row labels, table pane 512
dictionary keys 177
ordered collection values 360, 361
printer fonts 383
divide: message
currency entry field 141
float entry field 231
integer entry fields 260
dividerDoubleClicked: event
   header control (Win32 only) 245
dividerDoubleClickedIndex: event
   header control (Win32 only) 246
dividing
   See also computation part; divide:
      message; modulo operations
      / message 340
      // message 340
currency entry field values 141
float entry field values 231
integer entry field values 260
number holder values 340, 347
quotient:and: message
   computation 106
divisors, returning greatest common 343
DLL accessors
   See also C structures
   use with C structures 77
DLL filename property
   DLL accessor 189
DLL procedures, invoking 188
DLLEXMPL.PAR file (Windows) 192
DLLEXMPO.PAR file (OS/2) 193
DLLs (Dynamic Link Libraries)
   field correspondences with C structures 71
   passing string holder contents to, as
   parameters 476
doubleClicked event
   toolbar (Win32 only) 535
doubleClicked: event
   button list box 64
   list panes 276
   list view (Win32 only) 285
tree view (Win32 only) 570
doubleClickedCoordinate: event
   value set (OS/2 only) 587
doubleClickedIndex: event
   list view (Win32 only) 286
   value set (OS/2 only) 587
doubleClickedOnCell: event
table pane 504
doubleClickedOnColumn: event
table pane 504
doubleClickedOnRow: event
table pane 504
down: message 343
Drag drop... property
   combo box 99
   drop-down list 203
   list pane 279
   multiple choice list 316
   rich edit 437
   text pane 524
Drag with Alt key property
toolbar (Win32 only) 538
dragFormats message
   container (OS/2 only) 122
drawing modes, Icon Editor 636
drawn button 195
drawn buttons
   See also push buttons
   Default button property 421
drive: event
   disk accessor 181
driveName: message
disk accessor 183
drives, returning lists of known 183
drop-down lists
   See also combo boxes; entry fields; list
   panes; menus
   Btrieve field and data type 50
DROPDOWN.PAR file 204
Duration property
   speaker 450
Dynamic Data Exchange. See DDE Client part;
DDE Server part
Index

**E**
- Edit bitmap standard property 634
- Edit column labels property table pane 512
- Edit contents... property combo box 98
dial pane 163
drop-down list 202
list pane 279
multiple choice list 315
nested part 325
notebooks 333
part accessor 367
rich edit 436
sliders (OS/2 only) 446
table pane 513
text pane 524
value set (OS/2 only) 590
- Edit graphic property
  - graph pane 242
- Edit graphic... button 634
- Edit graphic... property
drawn button 197
static graphic 462
- Edit icon standard property 635
- Edit icon... button xxiv
- Edit menu
  - Bit Editor 635
  - Icon Editor 637
- Edit picture... property
  - picture entry field 379
- Edit property
  - property constructor 397
- Edit row labels property
  - table pane 512
- Edit value... property
  - array holder 41
dictionary holder 179
ordered collection holder 360
value holder 581
- Edit... button
  - DLL accessor 190
- Edit... property
  - master spin button (OS/2 only) 456
editData message
  - container (OS/2 only) 122
- editing
  - See also changing; direct-editing; property editing
  - items in ordered collections 360
  - nested part contents 322
  - PAR and PAX files 364
  - part icons xxiv
  - table pane cells 503
- Editor property
  - property constructor 397
editProperties message
  - OLE control (Win32 only) 352
editPropertyFile: message
  - property interface 405
element messages 37
  - element message 39
  - elements
    - array 39, 40
    - ordered collection holders 357, 358, 359
    - string holder 477
  - ellipsis (…)
    - number holders 339, 349
    - string holders 473, 478
  - embedded nested parts
    - editing 322
    - updating contents of 324
  - embedded parts
    - editing 322
    - updating 324
  - embedded reuse mode 319, 321
  - emphasizeInUse: message
    - container (OS/2 only) 122
  - empty strings
    - phone number entry fields 369, 370
    - picture entry fields 376
    - setting alphabetic entry fields to 26
enable message
  - alphabetic entry fields 26
  - button list box 66
  - check box 87
  - combo box 97
  - container (OS/2 only) 122
  - currency entry field 141
date entry field 147
dial pane 162
drawn button 196
drop-down list 201
entry field 212
float entry field 231
graph pane 240
header control (Win32 only) 246
integer entry fields 260
list panes 277
list view (Win32 only) 287
menu bitmap item 298
menu label 304
menu text item 308
multiple choice lists 313
notebooks 329
OLE control (Win32 only) 352
phone number entry fields 370
picture entry fields 377
push buttons 420
radio buttons 424
rich edit (Win32 only) 434
scroll bars (Win32 only) 440
sliders (OS/2 only) 444
spin buttons (OS/2 only) 454
static text 463
tab control (Win32 only) 487
table pane 507
text pane 521
toolbar (Win32 only) 536
toolbar button (Win32 only) 544
toolbar toggle button (Win32 only) 555
track bar 563
tree view (Win32 only) 572
value set (OS/2 only) 587
video pane (Win32 only) 594
Enable mini icons property
container (OS/2 only) 131
Enabled property
toolbar button (Win32 only) 546
toolbar toggle button (Win32 only) 557
enabling parts. See enable message
end event
track bar 561
end Adjust event
 toolbar (Win32 only) 535
endJob message
printer 382
endOfFileException event
Btrieve accessor 51
End-user help file property
window 608
ensemble reuse mode 320
ever: event
container (OS/2 only) 118
entered: event
alphabetic entry fields 25
combo box 96
currency entry field 140
date entry field 146
entry field 211
float entry field 230
integer entry fields 259
phone number entry fields 370
picture entry fields 376
ENTRXMPO.PAR file (OS/2 only) 215
ENTRXMPW.PAR file (Win32 only) 215
entry fields
See also entries for specific entry field
parts; combo boxes; drop-down lists; list panes; text panes
Btrieve field and data type 50
combo box 95
described 209
Entry point property
DLL accessor 190
error: event
printer 381
events
Btrieve accessor 59
C structure vs. DLL field alignment 71
conditional message execution 271
copying to clipboard 91
date entry field 146
directory operations 181, 182
tenvironmental fields, too many 211
executing DDE commands 158
file access 432, 520
file operation 217
float entry field values 230
graph pane 239
integer entry field 260
invalid currency entry field values 140
nonexistent dictionary holder keys 175
opening DLL files 187, 188
phone number entry field 370
picture entry field 377
printing process 381
typed value holder mismatches 577
evaluate: message
property interface 405
ever message 343
event/part cross-reference 615
events
alphabetic entry field 25
array holder 37
button list box 63
C structure 73
clipboard accessor 91
comparison 101
container (OS/2 only) 118
relevant to Btrieve record fields 49
DDE client 152
disk accessor 181
header control (Win32 only) 245
help menu 251
information dialog 255
key access 180
list view (Win32 only) 285
menu bitmap item 297
menu text item 307
printer part 381
progress bar 385
prompter 391
property constructor 396
property interface 404
radio button 424
rich edit (Win32 only) 432
status window (Win32 only) 467
tab control (Win32 only) 486
tab control page (Win32 only) 495
track bar 561
tree view (Win32 only) 570
exception: event
  Btrieve accessor 51
exceptionString: message
  Btrieve accessor 52
  converting Btrieve status codes to text 51
Exclude field button
  Btrieve Form Item Layout dialog 57
  C structure Form Item Layout dialog 77
EXE files
  including with delivered applications 364
  saving applications as 32
executeCommand: event
  DDE server 158
executeCommand: message
  DDE client 152
executionFailed message
  DDE server 158
exists: message
  disk accessor 183
  file accessor 219
triggering existsTrue: event 181
existsTrue: event
  disk accessor 181
  exists: message triggering 183
  file accessor 217
Exit (Bit Editor File menu) command 634, 637
exiting
  Bit Editor 634, 637
  Icon Editor 637
exp message 343
expanded: event
  tree view (Win32 only) 570
expandItem: event
  container (OS/2 only) 118
expandItem: message
  tree view (Win32 only) 572
expandTreeForItem: message
  container (OS/2 only) 122
explicit file loading 364
exponentiation messages 343, 347
extendedSelect message
  container (OS/2 only) 123
false event
  comparison 101
Fast spin property
  master spin button (OS/2 only) 455
field message
  Btrieve accessor 52
  C structure 74
Field name property
  C structure 75
Field Type property
  status window (Win32 only) 470
field type property
  C structure 76
field types
  Btrieve, vs. PARTS data types 58
  C structure 76
field: event
  Btrieve accessor 51
  C structure 73
field: message
  C structure 74
fieldChanged event
  container (OS/2 only) 118
fields
  aligning C structure with DLL 71
  C structure correspondences to DLL 71
  defining C structure 75
  editing Btrieve database 55
  lengths of Btrieve 58
  messages generated for Btrieve 49
  renaming C structure 77
  setting properties for Btrieve database 57
  size of C structure 76
Fields property
  Btrieve accessor 55
  C structure 75
fields: message
  container (OS/2 only) 123
file accessors. See also file dialogs 217
file extensions
  part accessors and 363
File menu
  Bit Editor 634
  Icon Editor 637
file menus. See also menus; menu text items;
  menu separators 227
File name property
  nested part 325
File Name... property
  AVI video (Win32 only) 47
  wave audio (Win32 only) 601
file referencing 363
FILEDIAL.PAR file 225
fileException: event
  file accessor 217
  graph pane 239
  rich edit (Win32 only) 432
  text pane 520
fileExtension: message
  file accessor 219
filename message
  AVI video (Win32 only) 44
  wave audio (Win32 only) 598
fileName: message
  file accessor 219
filename: message
  AVI video (Win32 only) 44
  wave audio (Win32 only) 598
Filename: property
  part accessor 367
fileNameException: event
  file accessor 217
filenames
  filtering 184, 185
  paths of 184
  returning
    lists of 183
    unqualified 219
files
  errors accessing 217, 218
  saving text pane contents to 434, 519, 522
files message
  disk accessor 183
files: event
  disk accessor 181
  setPattern: message triggering 184
filesIn: message
  disk accessor 183
filesIn:matching: message
  disk accessor 183
files:matching: message 183
fileText: message
  as argument for print: message 384
  file accessor 219
fileTextToThreshold: message
  file accessor 219
Fill (Bit Editor Edit menu) command 635
Fill (Icon Editor Edit menu) command 637
filtering filenames 184
  Default pattern property 225
  disk accessor Pattern property 185
  file dialogs 223
first message
  array holder 39
  ordered collection holders 357
  string holders 477

firstPage message
  notebooks 329
  tab control (Win32 only) 488
firstSelection message
  container (OS/2 only) 123
fixed or proportional attributes of Size window property
  646
fixed or proportional attributes, window layout
  646
float entry fields 229
  See also alphabetic entry fields; currency entry fields; date entry fields; entry fields; integer entry fields; phone number entry fields; picture entry fields; alphabetic entry fields and 29
  Btrieve field and data type 50
Float, class
  converting strings/numbers to 136
  currency entry field 141, 147
  currency entry fields 139, 231, 261
floating point numbers. See asFloat message; float entry fields
floor message. See also ceiling message 343
focus, input. See setFocus message; tabbed event
Font standard property 638
Font... property
  printer 383
fonts
  displaying printer 383
  selecting printer 382
Forecolor standard property 639
Form Item Layout dialog 76
Format property
  property constructor 398
formFeed message
  printer 382
fractions. See denominator message; numerator message
free message
  C structure 74
freeing
  memory 364, 367
  system resources 528
Frequency property
  speaker 450
frequency:duration: message
  speaker 449
G
gcd: message 343
Generate property dialog property
  property interface 407
Index

getBitmap message
  clipboard accessor 92
  getException event triggered by 91
  gotBitMap event triggered by 91
getEqual message
  Btrieve accessor 52
  getException event
    clipboard accessor 91
    clipboard accessor messages triggering 92
getFields event
  container (OS/2 only) 119
getFirst message
  Btrieve accessor 52
getGreaterOrEqual message
  Btrieve accessor 52
getIconFor: message
  container (OS/2 only) 123
ggetItemNamed: message
  container (OS/2 only) 123
ggetItemOf: message
  container (OS/2 only) 123
getLast message
  Btrieve accessor 52
getLessOrEqual message
  Btrieve accessor 52
getNameFor: message
  container (OS/2 only) 123
getNext message
  Btrieve accessor 53
goToObjectOf: message
  container (OS/2 only) 123
getPrevious message
  Btrieve accessor 53
ggetString message
  clipboard accessor 92
  getException event triggered by 91
  gotString: event triggered by 91
gotBitMap: event
  clipboard accessor 91
  getBitmap message triggering 92
gotString: event
  clipboard accessor 91
  getString message triggering 92
graph panes
  See also static graphics
  Btrieve field and data type 50
graphics. See bitmap files; graph panes; static graphics
greater than (>) message
  number holders 341
  string holders 474
greater than or equal (>=) message
  number holders 341
  string holders 474
greatest common divisor 343
Grid (Bit Editor Options menu) command 655
Grid (Icon Editor Option menu) command 637
Grouped property
  toolbar button (Win32 only) 546
  toolbar toggle button (Win32 only) 556
growTextLimit message
  rich edit (Win32 only) 434
GRPHXMPL.PAR file 242

H
headingChanged event
  container (OS/2 only) 119
headings message
  list view (Win32 only) 287
headings: message
  list view (Win32 only) 287
help file
  assigning to a window 250
help topics
  identifying 250
help: event
  tab control page (Win32 only) 496
helpManager: message
  windows 606
Hidden property
  toolbar button (Win32 only) 547
  toolbar toggle button (Win32 only) 557
hiddenIsTrue event
  menu bitmap item 297
  menu item separator 301
  menu text item 307
hide message
  menu bitmap item 298
  menu item separator 301
  menu text item 308
  toolbar (Win32 only) 536
  toolbar button (Win32 only) 544
  toolbar toggle button (Win32 only) 555
hideItem: message
  container (OS/2 only) 124
hideWindow message
  windows 606
higherOf: message 344
Hint property
  application 33
hints
  array holder values 42
  dictionary holder 180
  for application 33
home event
  track bar 561
horizontal attributes of Size window property
  646
horizontal attributes, window layout 646
Horizontal property
  track bar 566
Horizontal scroll bar property
  graph pane 241
  list pane 279
  multiple choice list 315
  table pane 512
  text pane 523
Hot link item events property
  DDE client 153
Hot link item messages property
  DDE server 160
hot-link item events
  DDE client property edit 153
  DDE conversations 157
  defining 157
hot-link item messages
  DDE conversations 157
  DDE server property edit 160
  defining 157
hotLink: event
  DDE client 152
hotLink: message
  DDE server 158

i
  .ico files 241
Icon Editor 461
  drawing modes 636
icon files
  setting graph panes to contents of 241
Icon property
  list view (Win32 only) 290
Icon property (OS/2 only)
  static graphic 462
Icon type property
  information dialog 257
icon types
  information dialog 257
icons
  See also static graphics
  application part and 31
  editing part xxiv
  loading from file 637
  minimizing windows to 605
  nested part, replacing application 322
Ignore buttons 113
Image property
  toolbar button (Win32 only) 547
  toolbar toggle button (Win32 only) 558
  includes: message
    array holder 39
    dictionary holder 176
    ordered collection holders 357
    string holders 477
  includesKey: message
    dictionary holder 177
  incrementPosition message
    progress bar 386
    track bar 563
  incrementPositionBy: message
    progress bar 386
    track bar 563
  incrementValue message 344
Indent property
  tree view (Win32 only) 573
Indeterminate property
  toolbar button (Win32 only) 547
  toolbar toggle button (Win32 only) 557
indeterminate: event
  button list box 64
indeterminateIndex: event
  button list box 64
indeterminateIndex: message
  button list box 66
indeterminateItem: message
  button list box 66
indexed access, table pane 512
indexes
  array holder element 37
  drop-down list items 200, 202
  list pane 278
  multiple choice list item 313
  table pane cell 501
  value set (OS/2 only) cells 585
indexException: event
  array holder 38
  at: message triggering 38
  at:put: message triggering 39
INFOXMP.Par file 258
Initial position property
  dialog window 173
  window 609
Initial setting property
  dial pane 164
  sliders (OS/2 only) 447
Initial size property
  window 609
Initial value property
  scroll bars (Win32 only) 441
Initial visibility property
  window 609
initializeObject: message
  property interface 405
initializing
  DDE conversations 156
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>number holders 349</td>
</tr>
<tr>
<td>value holders 578</td>
</tr>
<tr>
<td>Initially on property check box 88</td>
</tr>
<tr>
<td>radio buttons 426</td>
</tr>
<tr>
<td>initiate message</td>
</tr>
<tr>
<td>DDE client 152</td>
</tr>
<tr>
<td>DDE server 158</td>
</tr>
<tr>
<td>initiating DDE conversations 160</td>
</tr>
<tr>
<td>initiateException: topic: event</td>
</tr>
<tr>
<td>DDE client 152</td>
</tr>
<tr>
<td>initX: event</td>
</tr>
<tr>
<td>property interface 404</td>
</tr>
<tr>
<td>input focus. See setFocus message; tabbed event</td>
</tr>
<tr>
<td>inputOccurred event</td>
</tr>
<tr>
<td>tab control (Win32 only) 487</td>
</tr>
<tr>
<td>insert message</td>
</tr>
<tr>
<td>Btrieve accessor 53</td>
</tr>
<tr>
<td>Insert parameter reference property string template 483</td>
</tr>
<tr>
<td>insert: event</td>
</tr>
<tr>
<td>container (OS/2 only) 119</td>
</tr>
<tr>
<td>insert: message</td>
</tr>
<tr>
<td>rich edit (Win32 only) 434</td>
</tr>
<tr>
<td>text pane 521</td>
</tr>
<tr>
<td>insertFolder: message</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>insertFolder: parent: message</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>insertFolders: message</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>insertFolders: parent: message</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>inserting. See adding</td>
</tr>
<tr>
<td>insertItem: message</td>
</tr>
<tr>
<td>button list box 66</td>
</tr>
<tr>
<td>combo box 97</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>drop-down list 201</td>
</tr>
<tr>
<td>list panes 277</td>
</tr>
<tr>
<td>multiple choice lists 313</td>
</tr>
<tr>
<td>insertItem: parent: message</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>insertItems: message</td>
</tr>
<tr>
<td>container (OS/2 only) 124</td>
</tr>
<tr>
<td>insertItems: parent: message</td>
</tr>
<tr>
<td>container (OS/2 only) 125</td>
</tr>
<tr>
<td>insertPathName: message</td>
</tr>
<tr>
<td>container (OS/2 only) 125</td>
</tr>
<tr>
<td>insertPathNames: message</td>
</tr>
<tr>
<td>container (OS/2 only) 125</td>
</tr>
<tr>
<td>integer entry fields</td>
</tr>
<tr>
<td>See also alphabetic entry fields; date entry fields; entry fields; float entry fields; phone number entry fields</td>
</tr>
<tr>
<td>Btrieve field and data type 50</td>
</tr>
<tr>
<td>Interface... button 632</td>
</tr>
<tr>
<td>Interval in millisecond property</td>
</tr>
<tr>
<td>timer 528</td>
</tr>
<tr>
<td>Inverse drawing mode of Icon Editor 636</td>
</tr>
<tr>
<td>invoking DLL procedures 188</td>
</tr>
<tr>
<td>isEqualTo: message</td>
</tr>
<tr>
<td>comparison 101</td>
</tr>
<tr>
<td>is:greaterThan: message</td>
</tr>
<tr>
<td>comparison 101</td>
</tr>
<tr>
<td>is:greaterThanOrEqualTo: message</td>
</tr>
<tr>
<td>comparison 102</td>
</tr>
<tr>
<td>is:identicalTo: message</td>
</tr>
<tr>
<td>comparison 102</td>
</tr>
<tr>
<td>is:lessThan: message</td>
</tr>
<tr>
<td>comparison 102</td>
</tr>
<tr>
<td>is:lessThanOrEqualTo: message</td>
</tr>
<tr>
<td>comparison 102</td>
</tr>
<tr>
<td>is:notEqualTo: message</td>
</tr>
<tr>
<td>comparison 102</td>
</tr>
<tr>
<td>is:notIdenticalTo: message</td>
</tr>
<tr>
<td>isAbove: message</td>
</tr>
<tr>
<td>message 344</td>
</tr>
<tr>
<td>isAboveEqual: message</td>
</tr>
<tr>
<td>message 344</td>
</tr>
<tr>
<td>isBelow: message</td>
</tr>
<tr>
<td>message 344</td>
</tr>
<tr>
<td>is:belowEqual: message</td>
</tr>
<tr>
<td>message 344</td>
</tr>
<tr>
<td>is:Checked message</td>
</tr>
<tr>
<td>menu bitmap item 298</td>
</tr>
<tr>
<td>menu text item 308</td>
</tr>
<tr>
<td>is:DetailView message</td>
</tr>
<tr>
<td>container (OS/2 only) 125</td>
</tr>
<tr>
<td>is:Disabled message</td>
</tr>
<tr>
<td>menu bitmap item 298</td>
</tr>
<tr>
<td>menu text item 308</td>
</tr>
<tr>
<td>isDoorOpen message</td>
</tr>
<tr>
<td>CD audio (Win32 only) 80</td>
</tr>
<tr>
<td>isEmpty message</td>
</tr>
<tr>
<td>array holder 39</td>
</tr>
<tr>
<td>dictionary holder 177</td>
</tr>
<tr>
<td>ordered collection holders 358</td>
</tr>
<tr>
<td>string holders 477</td>
</tr>
<tr>
<td>isFirstPage message</td>
</tr>
<tr>
<td>tab control (Win32 only) 488</td>
</tr>
<tr>
<td>isHidden message</td>
</tr>
<tr>
<td>menu bitmap item 298</td>
</tr>
<tr>
<td>menu item separator 301</td>
</tr>
<tr>
<td>menu text item 308</td>
</tr>
<tr>
<td>isIconView message</td>
</tr>
<tr>
<td>container (OS/2 only) 125</td>
</tr>
</tbody>
</table>

666 Visual Smalltalk Enterprise Part Reference
isLastPage message
  tab control (Win32 only) 488
isLeftEqualOf: message 345
isLeftOf: message 345
isMediaPresent message
  CD audio (Win32 only) 80
isModified message
  rich edit (Win32 only) 434
text pane 521
isNameView message
  container (OS/2 only) 125
isNotReady message
  AVI video (Win32 only) 44
  CD audio (Win32 only) 80
  wave audio (Win32 only) 598
isOn message
  toolbar toggle button (Win32 only) 555
isOpen message
  AVI video (Win32 only) 44
  wave audio (Win32 only) 598
isPaused message
  AVI video (Win32 only) 44
  CD audio (Win32 only) 80
  wave audio (Win32 only) 598
isPlaying message
  AVI video (Win32 only) 44
  CD audio (Win32 only) 80
  wave audio (Win32 only) 598
isReady message
  AVI video (Win32 only) 44
  CD audio (Win32 only) 80
  wave audio (Win32 only) 598
isRightEqualOf: message 345
isRightOf: message 345
isSeeking message
  AVI video (Win32 only) 44
  CD audio (Win32 only) 81
  wave audio (Win32 only) 598
isSelected: message
  container (OS/2 only) 125
isStopped message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 81
  wave audio (Win32 only) 598
isTimeFormatBytes message
  wave audio (Win32 only) 599
isTimeFormatFrames message
  AVI video (Win32 only) 45
isTimeFormatMilliseconds message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 81
  wave audio (Win32 only) 599
isTimeFormatMsf message
  CD audio (Win32 only) 81
isTimeFormatSamples message
  wave audio (Win32 only) 599
isTimeFormatTmsf message
  CD audio (Win32 only) 81
isTreeView message
  container (OS/2 only) 125
Item Border property
  value set (OS/2 only) 589
items message
  toolbar (Win32 only) 536

K
key message
  automatic generation of 180
dictionary holder 177
Key: property
  dictionary holder 179
keyAtValue: message
  dictionary holder 177
keyboard operations, input focus and 640
keyException: event
  dictionary holder 175
  messages triggering 176, 178
keys
  Btrieve accessor 49, 58
  dictionary holder 175, 501
keys message
  dictionary holder 177
keys, shortcut. See shortcut keys
keysSorted message
  dictionary holder 177
knownDrives message
  disk accessor 183

L
Label left of button property
  radio button 426
Label left of check box property (Windows only)
  check box 88
label message
  check box 87
dialog window 170
  group pane 243
  menu label 304
  menu text item 308
  push buttons 420
  radio buttons 425
table pane 505
toolbar button (Win32 only) 545
toolbar toggle button (Win32 only) 555
windows 606
Index

Label property
  check box 88
  group pane 244
  menu label 304
  menu text item 309
  push button 421
  radio button 426
  static text 464
  toolbar button (Win32 only) 547
  toolbar toggle button (Win32 only) 558
labels
  See also label message; Label property;
  menu labels; setLabel: message; titles
  check box 85, 87, 88
  confirmation dialog button 111
  confirmation with cancel 116
  dialog window 172
  group pane 243
  push button 420, 422
  radio buttons 425
last message
  array holder 40
  ordered collection holders 358
  string holders 477
lastPage message
  notebooks 330
  tab control (Win32 only) 488
launch message
  launch pad 266
launch: message
  launch pad 266
launchException event
  launch pad 266
launchPartFile: message
  launch pad 266
layout attributes, part 644
lcm: message 345
least common multiple, returning 345
Left property
  list view (Win32 only) 291
left: message 345
leftMost: message 346
length message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 81
  wave audio (Win32 only) 599
lengthOfTrack: message
  CD audio (Win32 only) 81
less than (<) message
  number holders 341
  string holders 474
less than or equal (<=) message
  number holders 341
  string holders 474
licensing applications containing Btrieve
  accessors 58
Line Increment property
  progress bar 388
  track bar 566
lineIncrement message
  progress bar 386
  track bar 563
lineIncrement: message
  progress bar 386
  track bar 563
Lines At Root property
  tree view (Win32 only) 573
Lines property
  tree view (Win32 only) 573
link firing sequences
  examining 169
  links connected to triggered: event 269
linked parts
  editing 322
  nested part source file removal 326
  updating 323
linked reuse mode 319, 321
LINKJUNC.PAR file 273
links
  See link firing sequences; link junction
  part
List box always visible property
  combo box 98
list message
  button list box 66
  combo box 97
  container (OS/2 only) 126
  drop-down list 201
  list panes 277
  multiple choice lists 313
list panes
  See also combo boxes; drop-down lists;
  entry fields; list panes; menus; multiple
  choice lists
  Btrieve field and data type 50
  described 275
List property
  list view (Win32 only) 290
listValue message
  combo box 97
ln message
  container (OS/2 only) 126
load message
  loading parts explicitly 364
  part accessor 366
Load Option property
  part accessor 367
loaded event
  part accessor 366
loading parts 364, 366
loading icons from file 637
Location fixed, size fixed radio button of Size window property 644
lock message
  Btrieve accessor 53
log: message 346
logarithms 346
logical operations. See also logicalAnd:and:message; logicalOr:and:message; not:message 105
logicalAnd:and:message computation 105
logicalOr:and:message computation 105
lowerOf:message 346
LSPNXMPL.PAR file 281

M
magnitude values 101
Major tabs height property notebook 333
Major tabs property (OS/2) notebooks 332
Major tabs width property notebook 333
makeIndeterminate message
  toolbar button (Win32 only) 545
  toolbar toggle button (Win32 only) 555
manual conventions xxi
Manufacturer property
  application 33
master spin buttons (OS/2 only) described 453
matrixIn:message
  table pane 507
max: message 346
Maximize button property window 609
maximizing windows 605
maximum message
  progress bar 386
  track bar 563
Maximum number of characters property
  alphabetic entry fields 28
  entry field 213
Maximum number of digits property
  currency entry field 142
  integer entry field 262
Maximum property
  progress bar 388
  scroll bars (Win32 only) 441
  track bar 566
maximum: message
  progress bar 386
  track bar 563
memory, freeing
  unload message 364
  by unloading parts 367
Menu bar property
  window 608
menu item separators
  See also menu bitmap items; menu labels; menu text items; menus
  positioning 295
menu text items
  positioning 295
MENU.PAR file 296
menus
  described 293
  selecting from menu bars 605
message text, information dialog 257
message/part cross-reference 619
messages
  See also error messages
  adding to DLL accessors 189
  alphabetic entry field 26
  array holder 38
  button list box 65
  C structure 74
  check box 86
  clipboard accessor 91
  computation 105
  conditional 270, 273
  confirmer part 110
  container (OS/2 only) 120
  DDE client 152
  element 37
  header control (Win32 only) 246
  help menu 251
  key access 180
  list view (Win32 only) 287
  menu text item 308
  progress bar 386
  property constructor 396
  property interface 405
  rich edit (Win32 only) 433
  speaker 449
  status window (Win32 only) 468
  tab control (Win32 only) 487
  track bar 562
  tree view (Win32 only) 571
  video pane (Win32 only) 593
  wave audio (Win32 only) 597
min: message 346
Minimize button property window 608
minimizing windows 605
Minimum Height property
    status window (Win32 only) 471
minimum message
    progress bar 386
    track bar 563
Minimum property
    progress bar 388
    scroll bars (Win32 only) 441
    track bar 566
Minimum Tool Height property
    toolbar (Win32 only) 539
Minimum Tool Width property
    toolbar (Win32 only) 539
minimum: message
    progress bar 386
    track bar 563
minimum:maximum: message
    progress bar 387
    track bar 563
minimum:maximum: position: message
    progress bar 387
minimum:maximum:position: message
    track bar 564
Minor tabs height property
    notebook 333
Minor tabs width property
    notebook 333
mnemonicTyped event
    static text 463
modal dialogs
    dialog windows 167
modified flag
    entering or modifying text 519
    setting 435, 520, 522
modifiedIsTrue event
    rich edit (Win32 only) 432
    text pane 520
Modify individual ticks property
    dial pane 165
    sliders (OS/2 only) 447
Modify ticks by interval property
    dial pane 165
    sliders (OS/2 only) 447
modifying. See changing, direct-editing, property editing
modulo operations
    \ message 341
    modulo:and: message
        computation 106
    remainder:and: message
        computation 106
    modulo:and: message
        computation 106
mouse cursor, entry fields and 209
mouse operations
    input focus and 640
Multi line tabs property
    tab control (Win32 only) 490
MULTIPLE.PAR file 317
multiples, returning least common 345
multipleSelect message
    container (OS/2 only) 126
multiply: message
    currency entry fields 141
    float entry field 231
    integer entry fields 261
multiplying
    * message 340
    currency entry field values 141
    float entry field values 231
    integer entry field values 261
    number holder values 340
    product:and: message
        computation 106
N
    name message
        table pane 505
    nameChanged event
        container (OS/2 only) 119
    named access, table pane 512
    needsChildrenFor: event
        tree view (Win32 only) 570
    needsColumnsFor: event
        list view (Win32 only) 570
    needsContents event
        button list box 64
        container (OS/2 only) 119
        list view (Win32 only) 286
    needsHasChildrenFor: event
        tree view (Win32 only) 571
    needsImageFor: event
        button list box 64
        list view (Win32 only) 286
    needsSelectedImageFor: event
        tree view (Win32 only) 571
    needsSmallImageFor: event
        list view (Win32 only) 286
    needsStateFor: event
        button list box 64
        list view (Win32 only) 286
    needsStringFor: event
        button list box 65
list view (Win32 only) 286
tree view (Win32 only) 571
negated message 346
negative message 346
Nested part use option property
   nested part 325
nested parts
   See also part accessors; reusing application parts
   application parts as 34
   linking with applications 34
   menus in 294
   ordering tab order of children 640
   PAR files as 32
NESTEDPA.PAR file 326
New (Bit Editor File menu) command 634, 637
New value property
   array holder 41
   dictionary holder 179
   ordered collection holder 359
   value holder 580
next: event
   tab control page (Win32 only) 496
nextLine event
   track bar 562
nextPage event
   track bar 562
nextPage message
   notebooks 330
   tab control (Win32 only) 488
nil object
   conversion messages returning 137
   setting currency entry field values to 140
   testing for, in value holders 579
   uninitialized value holders 578
No adjust height property
   list pane 279
   multiple choice list 315
No border property
   master spin button (OS/2 only) 455
No button
   confirmation with cancel dialogs 113
   triggering no event 109
no event
   confirmer 110
   confirmer with cancel 114
   No or Cancel button presses triggering 109
non-automatic check boxes 85, 88
nonvisual parts
   direct-editing xxiii
   nested 319
   overview xxi
   property editing xxiv
   updating nested parts referencing 323
noSuchFileException: event
   file accessor 218
not: message
   computation 106
notebook pages. See also notebooks 327
NOTEXMPL.PAR file 334
null value
   phone number entry fields 369, 370
   picture entry fields 376
   setting alphabetic entry fields to 26
   number holders
   icons for xxiv
Number of columns property
   table pane 511
Number of rows property
   table pane 511
Number of ticks property
   dial pane 164
   sliders (OS/2 only) 447
numberOfTracks message
   CD audio (Win32 only) 81
numerator message 347
O
Object expression property
   property interface 406
objectAt: message
   container (OS/2 only) 126
objectsMatching: message
   container (OS/2 only) 126
objectUnderCursor message
   container (OS/2 only) 126
occurrencesOf: message
   array holder 40
   dictionary holder 177
   ordered collection holders 358
   string holders 477
odd message. See also even message 347
offsets, DLL field 72
OK button
   triggering yes event 109
OK/Cancel confirmation dialogs 109
ok: event
   prompter 391
Open as desktop child property (OS/2 only)
   dialog window 173
   window 609
Open in details view property
  container (OS/2 only) 133
open message
  application part 32
  AVI video (Win32 only) 45
  Btrieve accessor 53, 54
  CD audio (Win32 only) 81
  confirmer part 110
  confirmer with cancel 114
dialog window 171
  DLL accessor 188
  DLL files 187
  help menu 251
  information dialogs 256
  nested parts 324
  Wait for answer property and 167
  wave audio (Win32 only) 599
  windows 607
Open... (Bit Editor File menu) command 634, 637
openDoor message
  CD audio (Win32 only) 81
opened event
dialog window 170
  initializing value holders 578
  windows 606
openException: event
  DLL accessor 188
opening
  See also loading
  BMP files 634
  Btrieve databases 54
  confirmation dialogs 110
  confirmation with cancel dialogs 114
  DDE conversations 156
  DLL files 187, 188
  files 223
  information dialogs 256
  new Workbenches 32
  Save file dialogs 224
Opening styles... property
dialog window 173
  window 609
openOn: message
  Btrieve accessor 53
openText: message
  confirmer 110
  confirmer with cancel 114
  information dialogs 256
openTitle: message
  confirmer 110
  confirmer with cancel 114
  information dialogs 256
operationException: event
disk accessor 182
Option (Icon Editor) menu 637
Options menu (Bit Editor) 635
Order children standard property 640
Order: property
date entry field 149
ordered collections
  converting
    array holder contents to 38
    Dictionary values to 176
    string holder contents to 475
OS/2
  defining currency formatting 139
  defining date formatting 145
P
  packaged parts. See also application reuse
  modes; embedded packaged parts; linked
  packaged parts 34
  packing DLL data structures 72
  padding C structures 73
  Page buttons height property
    notebook 333
  Page buttons width property
    notebook 333
  Page Increment property
    track bar 566
  Page key property
    notebook page 336
  page message
    notebooks 330
  pageIncrement message
    track bar 564
  pageNumber message
    notebooks 330
    tab control (Win32 only) 488
PAR files
  See also PAX files
  applications containing nested parts 34
  part accessor references to 363
  PAX files vs. 364
  sample applications xxiv
  saving applications as 32
Parameter markers property
  string template 482
parameters
  passing string holder contents to DLL
  procedures 476
parent parts
  modifying tab order of children 640
parentheses ( )
  parameter markers 482
Part file reuse option property
  application 34
part files. See PAR files; PAX files
Part name standard property 643
part names, changing. See direct-editing
PARTACC.PAR file 368
Partial read threshold property
  file accessor 220
parts
  See also entries for specific parts; nonvisual
  parts; visual parts
  candidates for use with part accessors 365
  loading/unloading 364, 367
PARTS Workbench
  application parts and 31
  overview xxi
Password property
  alphabetic entry field 29
  currency entry field 143
  date entry field 149
  entry field 214
  float entry field 233
  integer entry field 263
  phone number entry field 372
  picture entry field 379
Paste (Bit Editor Edit menu) command 635
pasteFromClipboard message
  graph pane 240
pasteSelection message
  rich edit (Win32 only) 434
  text pane 522
pasting
  See also clipboard accessor
  bitmaps from clipboard 240
  entry field text to clipboard 210
  text 522
pathName message
  disk accessor 184
pathName: event
  disk accessor 182
  file dialog 223
pathNameForOpen message
  file dialog 223
pathNameForOpen: message
  file dialog 224
pathNameForSave message
  file dialog 224
pathNameForSave: message
  file dialog 224
pathNameOfFile: message
  disk accessor 184
pathNameOfFile:In: message
  disk accessor 184
pathnames
  changes triggering pathName: event 182
  directoryPath: message returning 183
  setting current directory 185
pattern message
  disk accessor 184
  file dialog 224
Pattern property
  disk accessor 185
pause message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 82
  wave audio (Win32 only) 599
PAX files
  See also PAR files
  part accessor references to 363
  referenced part file copies 34
  vs. PAR files 364
pen message
  graph pane 240
phone number entry fields
  See also alphabetic entry fields; currency
  entry fields; date entry fields; entry
  fields; float entry fields; integer entry
  fields; picture entry fields
  Btrieve field type 50
pixels
  defined 645
Place At Top property
  status window (Win32 only) 469
  toolbar (Win32 only) 538
play message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 82
  wave audio (Win32 only) 599
playCompleted event
  AVI video (Win32 only) 43
  CD audio (Win32 only) 79
  wave audio (Win32 only) 597
playFrom: message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 82
  wave audio (Win32 only) 599
playFrom:to: message
  AVI video (Win32 only) 45
  CD audio (Win32 only) 82
  wave audio (Win32 only) 599
points, table pane cells and 505, 506
Poke item events property 159
Poke item messages property
  DDE client 154
Index

pokeFailed message
 DDE server 158
poke-item events
 DDE Server part property edit 159
defining 157
poke-item messages
 DDE client property edit 154
defining 157
described 151
pokeItem: event
 DDE server 158
pokeItem: message
 DDE client 152
pokeItem:value: event
 DDE server 158
pokeItem:value: message
 DDE client 152
triggering pokeItem:value: event 158
pop-up menus 293
popUp message
 adding edit menus to text panes 525
 creating submenus 307
 menus 294
triggering 293
pop-up windows. See dialog windows
popupHelpMenu message
 help menu 251
position message
 CD audio (Win32 only) 82
 progress bar 387
 track bar 564
 wave audio (Win32 only) 599
Position property
 progress bar 388
 track bar 565
position: message
 AVI video (Win32 only) 45
 CD audio (Win32 only) 82
 progress bar 387
 track bar 564
 wave audio (Win32 only) 600
positionOfTrack: message
 CD audio (Win32 only) 82
positive message 347
#pragma pack directive 72
preventing dialog window closure 168
previousLine event
 track bar 562
previousPage event
 track bar 562
previousPage message
 notebooks 330
tab control (Win32 only) 488
print message
 rich edit (Win32 only) 434
print: message
 printer 382
Printer name property
 printer 383
printer parts 381
printerName message
 printer 382
printerNames message
 printer 382
printerSetup message
 printer 382
printScreen message
 printer 382
printUserRectangle message
 printer 382
PRNTXmpl.PAR file 384
procedureName message
 DLL accessor 188
Procedures property
 DLL accessor 189
procedures, invoking DLL 188
product:and: message
 computation 106
productInformation event
 help menu 251
prompt message
 prompter 391
Prompt property
 prompter 393
prompt:default: message
 prompter 392
promptDefault: message
 prompter 392
PROMPTER.PAR file 394
promptMessage message
 prompter 392
Properties dialogs, accessing xxiv
Properties property
 property constructor 397
Properties... property
 C structure 76
property edit
 toolbar (Win32 only) 537
 wave audio (Win32 only) 601
property editing
 alphabetic entry fields 28
 application parts 33
 array holder 40
Btrieve accessor 54
button list box 67
C structures 75
check boxes 88
combo box 98
confirm 111
confirm with cancel 115
currency entry field 142
DDE client 153
dial pane 163
dialog window 171
dictionary holder 178
DLL accessor 189
drawn button 197
drop-down list 202
entry field 213
file accessor 220
file dialog 225
float entry field 232
graph pane 241
group pane 244
header control (Win32 only) 247
integer entry fields 262
list view (Win32 only) 288
menu bitmap item 299
menu item separator 302
menu text item 309
multiple choice lists 315
nested parts 325
notebooks 331
number holders 349
ordered collection holders 359
overview xxiv
part accessors 367
phone number entry fields 372
printer parts 383
progress bar 388
property constructor 396
property interface 406
radio buttons 426
rich edit (Win32 only) 436
status window (Win32 only) 468
tab control (Win32 only) 489
text panes 523
toolbar toggle button 556
track bar 565
tree view (Win32 only) 573
Property editor file property
property constructor 397
proportional attributes, window layout 646
PSHBMPLPAR file 422
pull-down menus
described 294
selecting, from menu bars 605

Q
quo: message 347
quotient:and: message
computation 106

R
radiansToDegrees message 347
radio buttons
See also check boxes; group panes
RADIOBTN.PAR files 428
Ragged right property
tab control (Win32 only) 490
raisedTo: message 347
raisedToInteger: message 347
Range angles property
dial pane 164
Range property
dial pane 164
rational numbers. See denominator message;
numerator message
Read only property
alphabetic entry fields 29
currency entry field 143
date entry field 149
entry field 214
float entry field 233
integer entry field 262
master spin button (OS/2 only) 455
phone number entry field 372
picture entry field 379
rich edit (OS/2 only) 438
sliders (OS/2 only) 446
table pane 512
text pane 523
Read-only text property
container (OS/2 only) 131
records, database
defining Btrieve database 49
deleting Btrieve 51
endOfFileException event triggered by
missing 51
rectangles, printing 382
referencedWhileUnloaded event
part accessor 366
referencing applications with nested parts 321
Registration Key Name property
toolbar (Win32 only) 540
Registration Value Name property
toolbar (Win32 only) 540
registrationKeyName message
toolbar (Win32 only) 536
registrationKeyName: message
toolbar (Win32 only) 536
registrationValueName message toolbar (Win32 only) 537
registrationValueName: message toolbar (Win32 only) 537
rem: message 347
remainder:and: message computation 106
Remove Field property C structure 76
Remove leading and trailing blanks property prompter 353
removew: message ordered collection holders 358
removeAll message dictionary holder 177
ordered collection holders 358
removeAll: message ordered collection holders 358
removeBlanks message prompter 392
removeFirst message ordered collection holders 358
removeKey: message dictionary holder 178
removeLast message ordered collection holders 358
removing See also deleting blanks
from prompters 392, 393
from string holder contents 478
Btrieve database record fields 55
C structure fields 76
check marks from menu items 299
check marks from menu text items 307, 309
child parts from tab order 641
groups of parts 642
items from ordered collections 360
multiple choice list items 311, 312
ordered collection holder elements 358
rename: message file accessor 219
replaceControlChars message string holders 477
replaceControlCharsIn: message conversion 136
replacing rich edit text 434
replacing text pane text 521
Report property list view (Win32 only) 290
requestItem: message DDE client 152
Reset button (Bit Editor) 634
Reset button (Icon Editor) 636
reset event toolbar (Win32 only) 535
reset message AVI video (Win32 only) 46
CD audio (Win32 only) 82
wave audio (Win32 only) 600
resizedIndex:to: event header control (Win32 only) 246
resizing bitmaps 635
dial panes 161
drawn buttons 195
windows 605
resizingIndex:to: event header control (Win32 only) 246
restoring windows 605
resume message AVI video (Win32 only) 46
CD audio (Win32 only) 82
wave audio (Win32 only) 600
retrieving values. See value message
Retry button confirmation with cancel dialogs 113
triggering yes event 109
Retry/Cancel confirmation dialogs 109
retryChange message alphabetic entry fields 26
currency entry field 141
date entry field 147
entry field 212
float entry field 231
integer entry field 261
phone number entry field 371
picture entry field 377
Return type property DLL accessor 191
returning values. See value message
reuse modes, application 320
reusing application parts. See also nested parts 34
reverse message AVI video (Win32 only) 46
reverse: message AVI video (Win32 only) 46
reversed message array holder 40
ordered collection holders 358
string holders 478
Ribbon strip property sliders (OS/2 only) 446
right: message 348
rightClicked event adding edit menus to text panes 525
| alphabetic entry fields | button list box | check box | combo box | currency entry field | date entry field | dial pane | dialog window | drawn button | drop-down list | entry field | float entry field | graph pane | integer entry fields | list panes | multiple choice lists | notebook pages | notebooks | phone number entry fields | picture entry fields | push buttons | radio buttons | rich edit (Win32 only) | scroll bars (Win32 only) | sliders (OS/2 only) | spin buttons (OS/2 only) | status window (Win32 only) | tab control (Win32 only) | tab control page (Win32 only) | table pane | text pane | track bar | value set (OS/2 only) | windows | rightMost: message | ring message | speaker | rounded message | roundTo: message | Row label width property | table pane | rowAt: message | table pane | rowAt:put: message | table pane | rowEntered: event | table pane | Rows property | value set (OS/2 only) | rowSelected: event | table pane | runtime | changing | static graphics at 461 | static text contents at 465 | executable file use at 364 | setting printer names at 381 |

*S*

Sample: property | date entry field 149 | samplesPerSecond message | wave audio (Win32 only) 600 | Save (Bit Editor File menu) command 634, 637 | Save as... (Bit Editor File menu) command 634, 637 | Save file dialogs, opening 224 | saveToFile: message | rich edit (Win32 only) 434 | text pane 522 | saving | applications 32 | event values/message results 577 | files 224 | icons 637 | rich edit text changes to file 434 | text pane changes to file 519, 522 | Scale property | sliders (OS/2 only) 446 | Screen drawing mode of Icon Editor 636 | screen. See coordinates, screen; printScreen message | scroll bars | See also dial panes | for list panes 275 | graph pane 241 | multiple choice lists 311 | table pane 512 | text pane 523 | Scroll Button property | sliders (OS/2 only) 446 | SCROLLBA.PAR file 442 | scrolling | See also Auto scroll property | alphabetic entry fields 28 | button list box 63 | combo box selection lists 95 | currency entry fields 143 | drop-down lists 199 | entry field text 210 | list panes 275 | Select (Bit Editor Edit menu) command 635 | Select page | tab control (Win32 only) 492 |
Select page property
  tab control page (Win32 only) 498
selectAFont message
  printer 382
selectAll message
  alphabetic entry fields 27
  container (OS/2 only) 126
  currency entry field 141
  date entry field 147
  entry field 212
  float entry field 231
  integer entry fields 261
  phone number entry fields 371
  picture entry fields 377
  rich edit (Win32 only) 435
  text pane 522
selectAPrinter message
  printer 382
selected event
  notebook pages 335
  tab control page (Win32 only) 496
Selected/Edit interface... command 632
selected: event
  list view (Win32 only) 286
selectedCell message
  table pane 507
selectedCell: event
  table pane 504
selectedGroup message
  table pane 508
selectedGroup: event
  table pane 505
selectedIndex: event
  table pane 505
selectedObjects message
  container (OS/2 only) 126
selectedPoint: event
  table pane 505
selectedText message
  rich edit (Win32 only) 435
  text pane 522
selectedIndex: message
  multiple choice lists 313
selecting
  See also selectAll message
  entry field text 209
  float entry field characters 231
  fonts 382
  from drop-down list 199
  integer entry field characters 261
  menus from menu bars 605
  multiple choice list items 313
  phone number entry field characters 371
  picture entry field text 377
  printers 382
  table pane cells 502
Selection Style property
  container (OS/2 only) 131
selectItem: message
  container (OS/2 only) 126
  multiple choice lists 313
  tree view (Win32 only) 572
selectItems: message
  container (OS/2 only) 126
Separator: property
  date entry field 149
Set master... property
  slave spin button (OS/2 only) 458
Set, class
  converting
    array holder contents to 38
    dictionary holder contents to 176
    string holder values to members of 476
  displaying all dictionary keys 177
setBitmap: message
  drawn button 196
  graph pane 240
  menu bitmap item 299
setColumnLabels: message
  table pane 508
setColumnNames: message
  table pane 508
setColumnWidths: message
  table pane 509
setCommand: message
  launch pad 266
setDefaultMessage: message
  prompter 392
setDirectory: message
  disk accessor 184
  setting current directory 185
  triggering drive: event 181
setDuration: message
  speaker 449
setElementn: message
  array holder 40
setField: message
  Btrieve accessor 55
setFileText: message
  file accessor 220
setFocus message
  alphabetic entry fields 26, 27
  button list box 66
  check box 87
  combo box 97
  container (OS/2 only) 126
  currency entry field 141
date entry field 147
dial pane 162
drawn button 196
drop-down list 201
entry field 212
float entry field 231
graph pane 240
header control (Win32 only) 247
integer entry fields 261
list panes 277
list view (Win32 only) 287
multiple choice lists 313
notebooks 330
OLE control (Win32 only) 352
phone number entry fields 371
picture entry fields 377
push buttons 420
radio buttons 425
rich edit (Win32 only) 435
scroll bars (OS/2 only) 440
spin buttons (OS/2 only) 454
tab control (Win32 only) 488
tabbed event and tabbed list 86
table pane 509
text pane 522
tree view (Win32 only) 572
value set (OS/2 only) 588
video pane (Win32 only) 594

setFocusAtRow:column: message
  setFocusAtRow:column: message
table pane

setFrequency: message
  setFrequency: message
  speaker 449

setFromBitmapFile: message
  setFromBitmapFile: message
  graph pane 241

setFromFile: message
  setFromFile: message
  rich edit (Win32 only) 435
text pane 522

setFromIconFile: message
  setFromIconFile: message
  graph pane 241

setFromReference: message
  setFromReference: message
  C structure 74

setHelpManager: message
  setHelpManager: message
  help menu 251

setInterval: message
  setInterval: message
timer 527

setItems: message
  setItems: message
  toolbar (Win32 only) 537

setKey: message
  setKey: message
  automatic generation of dictionary holder 178

setLabel: message
  setLabel: message
  check box 87

dialog window 171
group pane 243
menu label 304
menu text item 308
push buttons 420
radio buttons 425	

setList: message
  setList: message
  button list box 66
  combo box 97
  container (OS/2 only) 127
drop-down list 201
list panes 277
multiple choice lists 314
spin buttons (OS/2 only) 454

setListOfPathNames: message
  setListOfPathNames: message
  container (OS/2 only) 127

setModified message
  setModified message
  rich edit (Win32 only) 435
text pane 522

setName: message
  setName: message
  table pane 508

setNumberOfColumns: message
  setNumberOfColumns: message
  table pane 509

setNumberOfRows: message
  setNumberOfRows: message
  table pane 509

setParameter: message
  setParameter: message
  string template 481

setPattern: message
  setPattern: message
disk accessor 184
  file dialog 224

setPrinterName message
  setPrinterName message
  printer 383

setPromptMessage: message
  setPromptMessage: message
  prompter 392

setRemoveBlanks: message
  setRemoveBlanks: message
  prompter 392

setRowLabels: message
  setRowLabels: message
  table pane 509

setRowNames: message
  setRowNames: message
  table pane 509

setTemplate: message
  setTemplate: message
  string template 481

setText: message
  setText: message
  confirmer 110
  confirmer with cancel 114
  information dialogs 256

setting
  setting
  initial value holder values. See also setValue: message 578
  modified flag 435, 522
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab order</td>
</tr>
<tr>
<td>with events and messages 642</td>
</tr>
<tr>
<td>value holder classes 578</td>
</tr>
<tr>
<td>setTitle: message</td>
</tr>
<tr>
<td>confirmer 111</td>
</tr>
<tr>
<td>information dialogs 256</td>
</tr>
<tr>
<td>setValue: message</td>
</tr>
<tr>
<td>alphabetic entry fields 27, 30</td>
</tr>
<tr>
<td>arguments triggering valueException:</td>
</tr>
<tr>
<td>event 577</td>
</tr>
<tr>
<td>button list box 66</td>
</tr>
<tr>
<td>combo box 97</td>
</tr>
<tr>
<td>converting array holder contents to</td>
</tr>
<tr>
<td>ordered collection parts 38</td>
</tr>
<tr>
<td>currency entry field 141</td>
</tr>
<tr>
<td>date entry field 147</td>
</tr>
<tr>
<td>dial pane 162</td>
</tr>
<tr>
<td>drop-down list 201</td>
</tr>
<tr>
<td>entering tab character into entry fields 214</td>
</tr>
<tr>
<td>entry field 212</td>
</tr>
<tr>
<td>float entry field 231</td>
</tr>
<tr>
<td>integer entry fields 261</td>
</tr>
<tr>
<td>list panes 277</td>
</tr>
<tr>
<td>multiple choice lists 314</td>
</tr>
<tr>
<td>phone number entry fields 371</td>
</tr>
<tr>
<td>picture entry fields 378</td>
</tr>
<tr>
<td>progress bar 387</td>
</tr>
<tr>
<td>radio buttons 425</td>
</tr>
<tr>
<td>rich edit (Win32 only) 435</td>
</tr>
<tr>
<td>scroll bars (Win32 only) 440</td>
</tr>
<tr>
<td>sliders (OS/2 only) 445</td>
</tr>
<tr>
<td>spin buttons (OS/2 only) 454</td>
</tr>
<tr>
<td>static graphics 461</td>
</tr>
<tr>
<td>static text 463</td>
</tr>
<tr>
<td>table pane 510</td>
</tr>
<tr>
<td>text pane 522</td>
</tr>
<tr>
<td>track bar 564</td>
</tr>
<tr>
<td>translating Dictionary to Array values 176</td>
</tr>
<tr>
<td>triggering checked event 86</td>
</tr>
<tr>
<td>value holders 579</td>
</tr>
<tr>
<td>setValueIndex: message</td>
</tr>
<tr>
<td>button list box 66</td>
</tr>
<tr>
<td>drop-down list 201</td>
</tr>
<tr>
<td>list panes 278</td>
</tr>
<tr>
<td>value set (OS/2 only) 588</td>
</tr>
<tr>
<td>setValueIndices: message</td>
</tr>
<tr>
<td>multiple choice lists 314</td>
</tr>
<tr>
<td>setWaitForAnswer: message dialog window 171</td>
</tr>
<tr>
<td>Containment 97</td>
</tr>
<tr>
<td>Size borders 608</td>
</tr>
<tr>
<td>Size in bytes property</td>
</tr>
<tr>
<td>C structure 76</td>
</tr>
<tr>
<td>size message</td>
</tr>
<tr>
<td>array holder 40</td>
</tr>
<tr>
<td>dictionary holder 178</td>
</tr>
<tr>
<td>string holders 478</td>
</tr>
</tbody>
</table>
Index

Size window... button 31
size: message
  file accessor 220
sizeException: event
  file accessor 218
sizes
  combo box 100
  initial window 609
slash (/) 145
slave spin buttons (OS/2 only)
  described 453
SLIDER.PAR file 448
Small Icon property
  list view (Win32 only) 290
Snap to increment property
  sliders (OS/2 only) 446
sortAscending: message
  list view (Win32 only) 287
sortByColumn message
  list view (Win32 only) 287
sortByColumn: message
  list view (Win32 only) 288
sortedAscending: message
  conversion 137
SortedCollection, class
  converting
    array holder contents to 38
    dictionary holder contents to 176
    ordered collection holder contents to 357
    string holder values to members of 476
  dictionary keys 177
sortedDescending: message
  conversion 137
sortItems message
  list view (Win32 only) 288
Spacing between ticks property
  sliders (OS/2 only) 447
speaker 449
speed message
  AVI video (Win32 only) 46
speed: message
  AVI video (Win32 only) 46
spin: message
  spin buttons (OS/2 only) 454
SPINBTN.PAR file 459
spinnedAround: event
  spin buttons (OS/2 only) 454
SPKRXMPL.PAR file 451
Split at pixel property
  container (OS/2 only) 133
splitColumn message
  container (OS/2 only) 127
splitColumn: message
  container (OS/2 only) 127
splitPosition message
  container (OS/2 only) 127
splitPosition: message
  container (OS/2 only) 127
sqrt message 348
square brackets [ ] 481, 482
standard part properties 631
  Backcolor 632
  Edit bitmap 634
  Edit icon 635
  Font 638
  Forecolor 639
  Order children 640
  Part name 643
  Size window 644
startTimer message
  timer 527
static graphics. See also graph panes 461
static text. See also group panes 463
STATICTE.PAR file 465
Status line text property
  notebook page 386
status line, of notebook 327
status lines, of notebook page 335
Status text alignment property
  notebooks 332
Status Window (Win32 only)
  described 467
stop message
  AVI video (Win32 only) 46
  CD audio (Win32 only) 83
  wave audio (Win32 only) 600
stopTimer message
  timer 528
stored event
  property interface 405
storePropertyValues message
  property interface 405
storing values. See value holders
Stretch bitmap to fit window property
  graph pane 242
Stretch property
  video pane (Win32 only) 595
string templates 481
String, returning members of class 136
strings
  array holder values as 41
  copying, cutting and pasting 91
  printing 381
  returned as clipboard contents 92
styles
   check box 88
   currency entry field 143
dialog window 172
Styles properties
   dialog window 172
Styles property
   alphabetic entry fields 28
   check box 88
   currency entry field 143
dial pane 163
subdirectories message
   disk accessor 184
subdirectories: event
   disk accessor 182
subdirectoriesIn: message
   disk accessor 184
substrings, creating 307
subtracting
   See also computation part
   - message 340
   currency entry field values 141
difference:and: message
   computation 105
   from float entry field values 232
   integer entry field values 261
   number holder values 340
sum:and: message
   computation 106
Support element messages and events property
   array holders 41
   dictionary holder 180
supportedEditors message
   property interface 405
supportedValidators message
   property interface 406
symbols
   converting strings to 476
   currency 139
System menu property (OS/2)
   window 608
T
Tab alignment property
   notebooks 332
Tab as button property
   tab control (Win32 only) 490
tab control (Win32 only) described 485
Tab fixed width property
   tab control (Win32 only) 490
Tab height property
   tab control (Win32 only) 491
Tab image property
   tab control page (Win32 only) 497
tab key, changing input focus using 640
Tab label property
   tab control page (Win32 only) 497
tab order
   automatic assignment of 640
   setting
      with events and messages 642
      setting with events and messages 642
tab order. See setFocus message; tabbed event
Tab property
   notebook page 336
Tab style property
   notebook page 336
Tab tip property
   tab control page (Win32 only) 497
Tab tips property
   tab control (Win32 only) 490
Tab width
   tab control (Win32 only) 491
tabbed event
   alphabetic entry fields 26
      button list box 65
      check box 86
      combo box 96
      currency entry field 140
date entry field 146
dial pane 162
drawn button 196
drop-down list 200
entry field 211
float entry field 230
graph pane 239
header control (Win32 only) 246
integer entry fields 260
list panes 276
list view (Win32 only) 286
multiple choice lists 312
notebooks 329
phone number entry fields 370
picture entry fields 376
push buttons 419
radio buttons 424
scroll bars (Win32 only) 440
sliders (OS/2 only) 444
spin buttons (OS/2 only) 454
tab control (Win32 only) 487
table pane 505
track bar 562
tree view (Win32 only) 571
value set (OS/2 only) 587
Tabs shape property (OS/2)
  notebooks 332
tabs, notebook page 327, 335, 336
tan message 348
TBPXMPL.PAR file 518
template message
  string template 482
Template property
  string template 483
terminate message
  DDE client 152
  DDE server 159
terminating
  DDE conversations 152, 159
Test (Icon Editor Option menu) command 637
text
  See also menu text items
    aligning
      alphabetic entry fields 29
      currency entry field 143
    confirmation dialog 109, 111
    confirmation with cancel dialogs 113, 114, 115
date entry field 148
  printing 381
  selecting currency entry field 141
Text align property
  master spin button (OS/2 only) 456
text cursor
  entry fields and 209
  moving within entry fields 210
text message
  conformer 111
  conformer with cancel 115
  information dialogs 256
text panes
  See also clipboard accessors; edit menus;
    entry fields
    Btrieve field types 50
Text property
  confirmation with cancel 116
  conformer 111
  conformer with cancel part 116
  information dialog 257
Three State property
  button list box 68
thresholds, file 219
 ticking event
timer 527
tickedTime: event
timer 527
Ticks on Both Sides property
  track bar 567
Ticks on Left/Top property
  track bar 567
Ticks on Right/Bottom property
  track bar 567
tilde (~)
  check box shortcut keys 89
  menu label shortcut keys 305
  notebook page shortcut keys 337
  push button shortcut keys 422
  radio button shortcut keys 427
  static text shortcut keys 465
time, returning current 527
time: message
  file accessor 220
timeFormat message
  AVI video (Win32 only) 46
  CD audio (Win32 only) 83
  wave audio (Win32 only) 600
timeFormat: message
  AVI video (Win32 only) 46
  CD audio (Win32 only) 83
  wave audio (Win32 only) 600
timeFormatBytes message
  wave audio (Win32 only) 600
timeFormatFrames message
  AVI video (Win32 only) 46
timeFormatMilliseconds message
  AVI video (Win32 only) 47
  CD audio (Win32 only) 83
  wave audio (Win32 only) 600
timeFormatMsf message
  CD audio (Win32 only) 83
timeFormatSamples message
  wave audio (Win32 only) 600
timeFormatTmsf message
  CD audio (Win32 only) 83
TIMRXMPL.PAR file 529
tip message
  toolbar button (Win32 only) 545
  toolbar toggle button (Win32 only) 555
Tip property
  toolbar button (Win32 only) 547
  toolbar toggle button (Win32 only) 558
Title bar property
  dialog window 172
  window 608
title message
  conformer 111
  conformer with cancel parts 115
  container (OS/2 only) 127
  information dialogs 256
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title property</td>
<td>application</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>confirmer</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>confirmer with cancel</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>container (OS/2 only)</td>
<td>132</td>
</tr>
<tr>
<td></td>
<td>information dialog</td>
<td>257</td>
</tr>
<tr>
<td>Title separator property</td>
<td>container (OS/2 only)</td>
<td>132</td>
</tr>
<tr>
<td>title</td>
<td>container (OS/2 only)</td>
<td>128</td>
</tr>
<tr>
<td>titleChanged event</td>
<td>container (OS/2 only)</td>
<td>119</td>
</tr>
<tr>
<td>titles</td>
<td>See also labels</td>
<td></td>
</tr>
<tr>
<td></td>
<td>application part</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>confirmation dialog</td>
<td>109, 111</td>
</tr>
<tr>
<td></td>
<td>confirmation with cancel</td>
<td>113, 114</td>
</tr>
<tr>
<td></td>
<td>dialog window</td>
<td>170, 171</td>
</tr>
<tr>
<td></td>
<td>information dialog</td>
<td>256, 257</td>
</tr>
<tr>
<td></td>
<td>window</td>
<td>607, 608</td>
</tr>
<tr>
<td>To</td>
<td>643</td>
<td></td>
</tr>
<tr>
<td>tone message</td>
<td>speaker</td>
<td>450</td>
</tr>
<tr>
<td>Tool Tip property</td>
<td>toolbar (Win32 only)</td>
<td>538</td>
</tr>
<tr>
<td>toolbar</td>
<td>adding to window</td>
<td>533</td>
</tr>
<tr>
<td></td>
<td>customizing</td>
<td>534</td>
</tr>
<tr>
<td></td>
<td>saving and restoring state</td>
<td>534</td>
</tr>
<tr>
<td>tools message</td>
<td>tool box (Win32 only)</td>
<td>531</td>
</tr>
<tr>
<td>Top property</td>
<td>list view (Win32 only)</td>
<td>291</td>
</tr>
<tr>
<td>Topic property</td>
<td>DDE client</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>DDE server</td>
<td>159</td>
</tr>
<tr>
<td></td>
<td>topics, DDE conversation</td>
<td>153</td>
</tr>
<tr>
<td>trigger message</td>
<td>link junction part</td>
<td>271</td>
</tr>
<tr>
<td>trigger</td>
<td>message</td>
<td></td>
</tr>
<tr>
<td>triggerApplyEvents message</td>
<td>tab control (Win32 only)</td>
<td>488</td>
</tr>
<tr>
<td>triggered: event</td>
<td>link junction part</td>
<td>271</td>
</tr>
<tr>
<td>triggerFailed: event</td>
<td>link junction part</td>
<td>271</td>
</tr>
<tr>
<td>triggerHelpEvent message</td>
<td>tab control (Win32 only)</td>
<td>489</td>
</tr>
<tr>
<td>triggerIfEmpty: message</td>
<td>link junction part</td>
<td>271</td>
</tr>
<tr>
<td>trigonometric function messages</td>
<td></td>
<td>341, 342, 347, 348</td>
</tr>
<tr>
<td>trimBlanks message</td>
<td>string holders</td>
<td>478</td>
</tr>
<tr>
<td>true event</td>
<td>comparison</td>
<td>101</td>
</tr>
<tr>
<td>truncated message</td>
<td></td>
<td>349</td>
</tr>
<tr>
<td>truncateTo: message</td>
<td></td>
<td>349</td>
</tr>
<tr>
<td>Turned On property</td>
<td>toolbar toggle button (Win32 only)</td>
<td>557</td>
</tr>
<tr>
<td>turnedOff event</td>
<td>radio buttons</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td>toolbar toggle button (Win32 only)</td>
<td>554</td>
</tr>
<tr>
<td>turnedOn event</td>
<td>radio buttons</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td>toolbar toggle button (Win32 only)</td>
<td>554</td>
</tr>
<tr>
<td>turnOff message</td>
<td>radio buttons</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>toolbar toggle button (Win32 only)</td>
<td>555</td>
</tr>
<tr>
<td>turnOn message</td>
<td>radio buttons</td>
<td>425</td>
</tr>
<tr>
<td></td>
<td>toolbar toggle button (Win32 only)</td>
<td>555</td>
</tr>
<tr>
<td>turnToPage: message</td>
<td>notebooks</td>
<td>330</td>
</tr>
<tr>
<td>turnToPageNumber: message</td>
<td>notebooks</td>
<td>330</td>
</tr>
<tr>
<td></td>
<td>tab control (Win32 only)</td>
<td>489</td>
</tr>
<tr>
<td>twips</td>
<td>645</td>
<td></td>
</tr>
<tr>
<td>TXTPXMPL.PAR file</td>
<td></td>
<td>526</td>
</tr>
<tr>
<td>typed value holders</td>
<td></td>
<td>577</td>
</tr>
</tbody>
</table>

**U**

uncheck message

check box 87

menu bitmap item 299

menu text item 309

unchecked event

check box 86

unchecked: event

button list box 65

uncheckedIndex: event

button list box 65

uncheckIndex: message

button list box 67
uncheckItem: message
  button list box 67
underscores (_): 375
undo message
  rich edit (Win32 only) 435
text pane 522
unformattedContents message
  rich edit (Win32 only) 435
unhideItem: message
  container (OS/2 only) 128
Unique key values property
  Btrieve accessor 58
Units of Measurement property
  unload message 645
unload message
  part accessor 364
unload event
  part accessor 366
unlock message
  Btrieve accessor 53
up: message 349
update message
  Btrieve accessor 53
Update object from file/expression property
  property interface 407
updateValue: message
  array holder 40
dictionary holder 178
updating
dictionaries 178
  nested parts from files 323
Use Images property
  button list box 68
list view (Win32 only) 289
tree view (Win32 only) 574
Use system settings property
date entry field 149
useFieldAsKey message
  Btrieve accessor 54

V

value holders
  See also array holders; dictionary holders;
  number holders; ordered collection
  holders; string holders
converting values in typed 137
value message
  alphabetic entry fields 27
button list box 67
C structure 74
check box 87
combo box 97
currency entry field 142
date entry field 148
dial pane 162
drop-down list 202
entry field 213
float entry field 232
integer entry fields 261
list panes 278
multiple choice lists 314
phone number entry fields 371
picture entry fields 378
progress bar 387
radio buttons 425
rich edit (Win32 only) 435
saving text pane changes 519
scroll bars (Win32 only) 440
sliders (OS/2 only) 445
spin buttons (OS/2 only) 455
static graphics 461
static text 464
string template 482
table pane 510
text pane 523
track bar 564
triggering ValuesIsOn event 86
value holders 579
Value property
  number holders 349
  ordered collection holder 360
  string holder 478
  value holder 580
Value when empty property
  alphabetic entry fields 28
  entry field 213
  picture entry field 379
value: message
  track bar 564
Value: property
dictionary holder 179
valueCoordinate message
  value set (OS/2 only) 588
valueCopy message
  value holders 579
valueException: event
  alphabetic entry fields 26
  array holder 38
  currency entry field 140
date entry field 146
  entry field 211
  integer entry fields 260
keyAtValue message triggering 177
phone number entry fields 370
picture entry fields 377
triggered by setValue: arguments 577
value holders 579
valueFormatted message
  alphabetic entry fields 27, 30
Index

currency entry field 142
  date entry field 148
  float entry field 232
  integer entry fields 261
  phone number entry fields 371
  picture entry fields 378
valueIndex message
  button list box 67
  drop-down list 202
  list panes 278
  value set (OS/2 only) 588
valueIndices message
  multiple choice lists 314
valuesEqual: message
  value holders 579
valuesNil: message
  value holders 579
valuesOn event
  check box 86
  radio buttons 424
valuesNotEqual: message
  value holders 580
valuesNotNil: message
  value holders 580
values, setting. See setValue: message
VALUESET.PAR file 591
variables. See value holders
version property
  application 33
vertical attributes of Size window property 646
vertical attributes, window layout 646
vertical property
  track bar 566
vertical scroll bar property
  graph pane 241
  table pane 512
  text pane 523
video message
  AVI video (Win32 only) 47
  video pane (Win32 only) 594
video: message
  AVI video (Win32 only) 47
  video pane (Win32 only) 594
view styles property
  container (OS/2 only) 132
viewDetails message
  container (OS/2 only) 128
viewIcon message
  container (OS/2 only) 128
viewing
  part properties 631
viewing application reuse modes 320
viewLarge message
  container (OS/2 only) 128
viewMini message
  container (OS/2 only) 128
viewName message
  container (OS/2 only) 128
viewNameFlow message
  container (OS/2 only) 128
viewNonDetails message
  container (OS/2 only) 128
viewNormal message
  container (OS/2 only) 128
viewText message
  container (OS/2 only) 129
viewTextFlow message
  container (OS/2 only) 129
viewTree message
  container (OS/2 only) 129
viewTreeIcon message
  container (OS/2 only) 129
viewTreeName message
  container (OS/2 only) 129
visual parts
  application part and 31
  C structure 77
  creating Btrieve accessor 57
  defining Btrieve fields based on 55
  direct-editing xxiii
  generating Btrieve database fields 50
  matching Btrieve database fields 49
  nested 319, 322
  overview xx
  property editing xxiv
  updating xxii
  updating nested parts referencing 323
W

Wait for answer property 112
  confirmer 112
  dialog window 173
  file dialog 225
  information dialog 258
  prompter 393
Wait for answer property (OS/2 only)
  confirmer with cancel 116
waitForAnswer message
  dialog window 171
  file dialog 224
  information dialogs 256
  prompter 392
waitForAnswer message (OS/2 only)
  confirmer with cancel 115
warning beep feature (OS/2) 255
width property
  status window (Win32 only) 470
  toolbar separator (Win32 only) 550
wildcards 184
Window label property
  window 608
WINDOW.PAR file 610
Windows
  defining currency formatting 139
  defining date formatting 145
windows
  See also dialog windows
  application 31
  default labels 33
  described 603
  dialog 167
  nested parts containing 326
Windows, Microsoft
  confirmation dialogs 112
  confirmation with cancel dialogs 116
WindowTitle event
  property interface 405
wizard
  adding Help button 612
  changing pages while building 612
  standard buttons 611
Word wrap property
  text pane 523
Wrapable property
  toolbar (Win32 only) 538
Wrapped property
  toolbar button (Win32 only) 546
  toolbar toggle button (Win32 only) 556
write message
  Btrieve accessor 54

Y
Year: property
  date entry field 149
Yes button
  confirmation with cancel dialogs 113
  triggering yes event 109
yes event
  confirmer 110
  confirmer with cancel 114
  YES, OK, Retry button presses triggering 109
Yes/No confirmation dialogs 109
Yes/No/Cancel confirmation dialogs 113
Reader Comment Sheet

Name: 

Job title/function: 

Company name: 

Address: 

Telephone number: (       )           Date:       /       / 

How often do you use this product? □ Daily □ Weekly □ Monthly □ Less

How long have you been using this product? □ Months □ Years

Can you find the information you need? □ Yes □ No

Please comment. 

Is the information easy to understand? □ Yes □ No

Please comment. 

Is the information adequate to perform your task? □ Yes □ No

Please comment. 

General comment: 

__________________________________________________________________

WE STRIVE FOR QUALITY

To respond, please fax to Larry Fasse at (513) 612-2000.