TN Bridge Host Integration Pack

Developer's guide
# Table of Contents

**Introduction** 27

1 Overview ........................................................................................................... 27
2 Architecture ...................................................................................................... 28
3 What's New in 3.5? ........................................................................................... 28
4 Trial & Full License .......................................................................................... 29

**The Development Lab** 30

1 Development Lab components ........................................................................... 31
   Main Menu and Toolbar ..................................................................................... 32
      Main Menu ...................................................................................................... 32
      Main Toolbar ................................................................................................. 35
   The Display workspace ....................................................................................... 36
      Interactive emulation ...................................................................................... 36
      Screen Snapshot ............................................................................................ 37
      XML views ..................................................................................................... 38
      Scripting window ............................................................................................ 38
   The Trace Event views ....................................................................................... 39
      Tree View ....................................................................................................... 39
      List View ........................................................................................................ 41
   The Object Inspector .......................................................................................... 41

2 Working with TN BRIDGE Development Lab .................................................. 42
   Development Lab modes .................................................................................... 42
      Interactive ...................................................................................................... 43
      Creating a connection .................................................................................... 43
      Connecting to the mainframe ........................................................................ 45
      Using XML Connections ............................................................................... 45
         Saving screens as XML files ................................................................... 46
         Connecting to a set of XML screen files ............................................... 47
      On-Line tracing .............................................................................................. 49
         Connecting to a TN BRIDGE Application .............................................. 50
      Off-Line tracing ............................................................................................. 51
         Opening a Trace File .................................................................................. 52
   Scripting ............................................................................................................ 52
   Working with XML ............................................................................................. 53
      Generic XML ................................................................................................. 54
      XML Templates ............................................................................................. 56
      Enabling Template Mode ............................................................................... 57
      Building XML Templates ............................................................................. 57
         Generating XML Template Fields ............................................................ 58
         Using Object Inspector ............................................................................ 60
         Identifying the screen ............................................................................... 61
         Deleting Template Fields ......................................................................... 61
         Saving an XML Template ....................................................................... 62
         Adding XML Template Field .................................................................. 63
         Adding XML Template Table .................................................................. 64
         Adding custom XML Template Areas ..................................................... 67

**Host Integration Pack Programming** 68

1 Host Integration Pack's component layers ....................................................... 68
Profiles ................................................................. 105
Port ......................................................................... 105
Rows ........................................................................ 106
ScreenRow .............................................................. 106
SessionId ............................................................... 106
SessionState .......................................................... 107
SSL .......................................................................... 107
SSLAcceptExpired .................................................. 107
SSLAcceptInValid .................................................... 108
SSLAcceptInValidCA .............................................. 108
SSLAcceptNotYetValid .......................................... 108
SSLAcceptSelfSigned ............................................. 109
SSLCertFile .......................................................... 109
SSLCertPassword ................................................... 109
SSLDelay .............................................................. 110
SSLEnableDialogs .................................................. 110
SSLLock ............................................................... 110
SSLMgmt .............................................................. 110
SSLRootCertFile .................................................... 111
SubKey ................................................................. 111
SynchronizeEvents ............................................... 111
TerminalType ....................................................... 112
UserId ................................................................. 112
XLockDelay .......................................................... 113
Methods .................................................................. 113
AsVariant .................................................................. 113
ClassName ................................................................ 113
CloseEditor .......................................................... 114
Connect .................................................................... 114
Disconnect ............................................................ 114
GetScreenRowEx ................................................... 115
GetScreenText ....................................................... 115
IsConnected .......................................................... 115
LoadFromXMLFile ............................................... 116
SaveToXMLFile .................................................... 117
SendAid .................................................................. 117
SendKeys ................................................................ 118
ShowEditor ........................................................... 119
Events ..................................................................... 120
OnConnect ............................................................ 120
OnConnectionFail ................................................. 120
OnDisconnect ....................................................... 120
OnMessageWaitingOff .......................................... 120
OnMessageWaitingOn ............................................ 120
OnScreenChange .................................................. 120
OnSendAid ............................................................ 120
OnSessionState ..................................................... 120
OnSystemLock ...................................................... 121
OnSystemUnlock ................................................... 121
Tnb3287XReference ............................................... 121
Tnb3287XControl ................................................... 121
Properties ............................................................. 122
AttachMode .......................................................... 122
CodePage .............................................................. 122

© 2015, Cybele Software, Inc.
Contents

ConversionTable.................................................................................................................. 123
Debug ................................................................................................................................. 124
DebugDir ............................................................................................................................ 125
DeviceName ....................................................................................................................... 125
Host .................................................................................................................................. 126
IsConnected ...................................................................................................................... 126
Port .................................................................................................................................... 126
PrinterCfg .......................................................................................................................... 127
ProfilesControl .................................................................................................................. 127
SendToPrinter .................................................................................................................... 127
SubKey ............................................................................................................................... 128
Text ..................................................................................................................................... 128
Methods ............................................................................................................................. 129
AboutBox ........................................................................................................................... 129
Connect .............................................................................................................................. 129
Disconnect ......................................................................................................................... 129
ShowPrinterEditor ............................................................................................................ 130
Events ................................................................................................................................ 130
OnConnect ........................................................................................................................ 130
OnConnectionFail ............................................................................................................. 130
OnDataAvailable ............................................................................................................. 130
OnDisconnect ................................................................................................................... 131
OnEndDocument ............................................................................................................... 131
OnStartDocument .............................................................................................................. 131

Tnb3812X Reference ........................................................................................................ 131

Tnb3812X Control .............................................................................................................. 131
Properties ......................................................................................................................... 132
CodePage ........................................................................................................................... 132
ConversionTable .............................................................................................................. 133
Debug ................................................................................................................................ 134
DebugDir ........................................................................................................................... 134
DeviceName ....................................................................................................................... 135
Host .................................................................................................................................. 135
IsConnected ...................................................................................................................... 136
MsgQueueLib .................................................................................................................... 136
MsgQueueName ............................................................................................................... 136
Port .................................................................................................................................... 137
PrinterCfg ........................................................................................................................ 137
ProfilesControl ................................................................................................................. 137
SendToPrinter .................................................................................................................... 137
SubKey ................................................................................................................................ 138
Text ..................................................................................................................................... 139
Methods ............................................................................................................................. 139
AboutBox ........................................................................................................................... 139
Connect .............................................................................................................................. 139
Disconnect ......................................................................................................................... 140
ShowPrinterEditor ............................................................................................................ 140
Events ................................................................................................................................ 140
OnConnect ........................................................................................................................ 141
OnConnectionFail ............................................................................................................. 141
OnDataAvailable ............................................................................................................. 141
OnDisconnect ................................................................................................................... 141
OnEndDocument ............................................................................................................... 141
OnStartDocument .............................................................................................................. 141

© 2015, Cybele Software, Inc.
TN BX POOL Reference ......................................................................................................................... 142
TN BX POOL Class ................................................................................................................................. 142
Properties .............................................................................................................................................. 142
AcquiredCount....................................................................................................................................... 142
Count....................................................................................................................................................... 142
Id ............................................................................................................................................................. 142
ReleasedCount....................................................................................................................................... 143
Size.......................................................................................................................................................... 143
Timeout................................................................................................................................................... 143
type........................................................................................................................................................ 143
Methods.................................................................................................................................................. 144
Acquire.................................................................................................................................................... 144
Release ................................................................................................................................................... 144
ReleaseAll .............................................................................................................................................. 144
TN BX FIELD Class Reference .................................................................................................................. 145
TN BX FIELD Class ................................................................................................................................. 145
Properties.............................................................................................................................................. 145
Attr .......................................................................................................................................................... 145
AutoEnter .............................................................................................................................................. 146
Blinking.................................................................................................................................................. 146
ByPass .................................................................................................................................................... 146
Col ........................................................................................................................................................... 147
Color ...................................................................................................................................................... 147
data ....................................................................................................................................................... 147
Eab .......................................................................................................................................................... 147
Editable .................................................................................................................................................. 148
High ......................................................................................................................................................... 148
Len ........................................................................................................................................................ 149
Mandatory............................................................................................................................................... 149
Modified ................................................................................................................................................ 149
MonoCase............................................................................................................................................. 150
Name ...................................................................................................................................................... 150
Numeric.................................................................................................................................................. 150
Pos ........................................................................................................................................................... 151
Reverse................................................................................................................................................... 151
Row ........................................................................................................................................................ 151
Skip ......................................................................................................................................................... 152
Underline............................................................................................................................................... 152
Visible.................................................................................................................................................... 153
HiAPIAttr ............................................................................................................................................... 153
HiAPIEab ................................................................................................................................................ 153
TN BX Fields Class Reference .................................................................................................................. 154
TN BX Fields Class ................................................................................................................................. 154
Properties.............................................................................................................................................. 154
Count ....................................................................................................................................................... 154
CursorField........................................................................................................................................... 154
CursorPos............................................................................................................................................. 155
Items ....................................................................................................................................................... 155
Methods.................................................................................................................................................. 156
Lock......................................................................................................................................................... 156
TnbXLPD Reference .............................................................................................................................. 156
TnbXLPD Control.................................................................................................................................. 156
Properties.............................................................................................................................................. 157
Active ...................................................................................................................................................... 157
Contents

CodePage ................................................................. 157
ConversionTable ....................................................... 158
CurrentQueue .......................................................... 159
Debug ................................................................. 159
DebugDir .............................................................. 159
DeleteFileAfterPrint .................................................. 160
Ebcdic ................................................................. 160
Enabled ............................................................... 160
Name ................................................................. 161
Port ................................................................. 161
PrintBanner ............................................................ 161
PrintDocument ........................................................ 161
QueueEnabled ......................................................... 162
QueueName ............................................................ 162
QueuePrinter ............................................................ 162
Queues ............................................................ 163
SpoolDir .............................................................. 163
Tag ................................................................. 164
UseFormFeedChar ...................................................... 164
Methods ............................................................. 164
AboutBox ............................................................. 164
CreateQueue .......................................................... 164
DeleteQueue ........................................................... 165
GetControlHandle .................................................... 165
LoadFromFile .......................................................... 166
SaveToFile ............................................................ 166
Events ............................................................... 166
OnConnect ............................................................ 166
OnDisconnect ........................................................ 167
OnEndControlFile .................................................... 167
OnEndDataFile ........................................................ 168
OnPrintBanner ........................................................ 168
OnPrintDocument ..................................................... 168
OnReceiveControlFile .............................................. 169
OnReceiveDataFile .................................................... 169
OnStartControlFile ................................................... 170
OnStartDataFile ....................................................... 170
TnbXLPDJob Reference ............................................... 171
TnbXLPDJob Class ...................................................... 171
Properties ............................................................ 171
BannerClass ........................................................... 171
BannerName ........................................................... 172
DataFileLength ........................................................ 172
DataFileName ........................................................ 173
DataFileReceived ..................................................... 173
ControlFileLength ..................................................... 173
ControlFileName ...................................................... 174
ControlFileReceived .................................................. 174
HostName ............................................................ 174
IsConnected .......................................................... 175
PrintMode ............................................................ 175
Queue ............................................................... 176
QueueName ........................................................... 176
SourceFileName ....................................................... 176

© 2015, Cybele Software, Inc.
<table>
<thead>
<tr>
<th>Property/Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>177</td>
</tr>
<tr>
<td>UserName</td>
<td>177</td>
</tr>
<tr>
<td>Constants</td>
<td>178</td>
</tr>
<tr>
<td>TxPrinterPrintMode</td>
<td>178</td>
</tr>
<tr>
<td>TxTnbLPDStatus</td>
<td>178</td>
</tr>
<tr>
<td>TnbXLPDJobs Reference</td>
<td>178</td>
</tr>
<tr>
<td>TnbXLPDJobs Collection</td>
<td>178</td>
</tr>
<tr>
<td>Properties</td>
<td>179</td>
</tr>
<tr>
<td>Items</td>
<td>179</td>
</tr>
<tr>
<td>Methods</td>
<td>179</td>
</tr>
<tr>
<td>Count</td>
<td>179</td>
</tr>
<tr>
<td>IndexOf</td>
<td>179</td>
</tr>
<tr>
<td>TnbXLPDQueue Reference</td>
<td>180</td>
</tr>
<tr>
<td>TnbXLPDQueue Class</td>
<td>180</td>
</tr>
<tr>
<td>Properties</td>
<td>180</td>
</tr>
<tr>
<td>CurrentJob</td>
<td>180</td>
</tr>
<tr>
<td>DeleteFileAfterPrint</td>
<td>181</td>
</tr>
<tr>
<td>Ebcdic</td>
<td>181</td>
</tr>
<tr>
<td>Enabled</td>
<td>181</td>
</tr>
<tr>
<td>Jobs</td>
<td>182</td>
</tr>
<tr>
<td>LPD</td>
<td>182</td>
</tr>
<tr>
<td>Name</td>
<td>183</td>
</tr>
<tr>
<td>PrintBanner</td>
<td>183</td>
</tr>
<tr>
<td>PrintDocument</td>
<td>183</td>
</tr>
<tr>
<td>SpoolDir</td>
<td>184</td>
</tr>
<tr>
<td>Methods</td>
<td>184</td>
</tr>
<tr>
<td>GetControlHandle</td>
<td>184</td>
</tr>
<tr>
<td>ShowPrinterEditor</td>
<td>184</td>
</tr>
<tr>
<td>Constants</td>
<td>185</td>
</tr>
<tr>
<td>TxBnbXLPDDelete</td>
<td>185</td>
</tr>
<tr>
<td>TnbXLPDQueues Reference</td>
<td>185</td>
</tr>
<tr>
<td>TnbXLPDQueues Collection</td>
<td>185</td>
</tr>
<tr>
<td>Properties</td>
<td>185</td>
</tr>
<tr>
<td>Items</td>
<td>185</td>
</tr>
<tr>
<td>Count</td>
<td>186</td>
</tr>
<tr>
<td>TNBXIndFile Reference</td>
<td>186</td>
</tr>
<tr>
<td>TNBXIndFile Control</td>
<td>186</td>
</tr>
<tr>
<td>Properties</td>
<td>187</td>
</tr>
<tr>
<td>AbortString</td>
<td>187</td>
</tr>
<tr>
<td>Append</td>
<td>187</td>
</tr>
<tr>
<td>Ascii</td>
<td>188</td>
</tr>
<tr>
<td>BlkSize</td>
<td>188</td>
</tr>
<tr>
<td>BufSize</td>
<td>189</td>
</tr>
<tr>
<td>Bytes</td>
<td>189</td>
</tr>
<tr>
<td>CRLF</td>
<td>190</td>
</tr>
<tr>
<td>Direction</td>
<td>190</td>
</tr>
<tr>
<td>FtCommand</td>
<td>190</td>
</tr>
<tr>
<td>FtMode</td>
<td>191</td>
</tr>
<tr>
<td>FtName</td>
<td>191</td>
</tr>
<tr>
<td>HostFileName</td>
<td>192</td>
</tr>
<tr>
<td>HostKind</td>
<td>192</td>
</tr>
<tr>
<td>LocalFileName</td>
<td>193</td>
</tr>
<tr>
<td>LRecl</td>
<td>193</td>
</tr>
<tr>
<td>PrimSpace</td>
<td>194</td>
</tr>
</tbody>
</table>
## Terminal Emulator Controls

<table>
<thead>
<tr>
<th>Constants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TNBXEmulatorControl</td>
<td>206</td>
</tr>
<tr>
<td>TNBXEmulatorControl</td>
<td>206</td>
</tr>
<tr>
<td>Properties</td>
<td></td>
</tr>
<tr>
<td>AidHookControl</td>
<td>207</td>
</tr>
<tr>
<td>Ctl3d</td>
<td>208</td>
</tr>
<tr>
<td>CursorPos</td>
<td>208</td>
</tr>
<tr>
<td>DisableKbdLockOnError</td>
<td>209</td>
</tr>
<tr>
<td>EnableSelection</td>
<td>209</td>
</tr>
<tr>
<td>ErrorLine</td>
<td>209</td>
</tr>
<tr>
<td>FontAutoSize</td>
<td>210</td>
</tr>
<tr>
<td>HotSpotsControl</td>
<td>210</td>
</tr>
<tr>
<td>InputInhibitEnabled</td>
<td>210</td>
</tr>
<tr>
<td>InsertMode</td>
<td>211</td>
</tr>
<tr>
<td>IsInputInhibited</td>
<td>211</td>
</tr>
<tr>
<td>LeftMargin</td>
<td>212</td>
</tr>
<tr>
<td>Profiles</td>
<td>212</td>
</tr>
<tr>
<td>RulerType</td>
<td>212</td>
</tr>
<tr>
<td>StyleName</td>
<td>213</td>
</tr>
<tr>
<td>SubKey</td>
<td>213</td>
</tr>
</tbody>
</table>

© 2015, Cybele Software, Inc.
Font ........................................................................................................230
TelnetControl .........................................................................................230
Visible ....................................................................................................231
Methods ...............................................................................................231
AboutBox ...............................................................................................231
AsVariant ...............................................................................................231
TNBXMacro Control ...............................................................................231
TNBXMacro Control...............................................................................231
Properties .............................................................................................232
AutoStartMacro .....................................................................................232
MacroName ............................................................................................232
ProfilesControl .....................................................................................232
State .......................................................................................................233
SubKey ....................................................................................................234
TelnetControl ........................................................................................234
Methods ...............................................................................................234
AboutBox ...............................................................................................234
AsVariant ...............................................................................................235
LoadFromXMLFile ................................................................................235
SaveToXMLFile .....................................................................................235
StartPlaying ...........................................................................................235
StartRecording .......................................................................................236
StopPlaying .............................................................................................236
StopRecording .........................................................................................237
Events ....................................................................................................237
OnStartPlaying .......................................................................................237
OnStartRecording ..................................................................................237
OnStopPlaying .........................................................................................237
OnStopRecording ....................................................................................237
Constants ..............................................................................................238
TNBMacroState ......................................................................................238
TNBXProfiles Control ..............................................................................238
TNBXProfiles Control ..............................................................................238
Properties .............................................................................................238
FileName ...............................................................................................238
Items .......................................................................................................239
Key .........................................................................................................239
ReadOnly ...............................................................................................240
StreamingType .......................................................................................240
Methods ...............................................................................................240
AboutBox ...............................................................................................240
AsVariant ...............................................................................................241
Commit ....................................................................................................241
Delete ......................................................................................................241
GetCurrent .............................................................................................242
GetXMLHeader .....................................................................................242
Rename ....................................................................................................242
Rollback .................................................................................................243
SetCurrent ..............................................................................................243
TNBXHotSpots Class Reference ..............................................................243
TNBXHotSpots Class Reference ..............................................................243
Properties .............................................................................................244
ActionFieldData .....................................................................................244
Active .....................................................................................................245
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>245</td>
</tr>
<tr>
<td>Description</td>
<td>245</td>
</tr>
<tr>
<td>EndCol</td>
<td>246</td>
</tr>
<tr>
<td>EndRow</td>
<td>246</td>
</tr>
<tr>
<td>Id</td>
<td>247</td>
</tr>
<tr>
<td>Pattern</td>
<td>247</td>
</tr>
<tr>
<td>StartCol</td>
<td>248</td>
</tr>
<tr>
<td>StartRow</td>
<td>248</td>
</tr>
<tr>
<td>TextColor</td>
<td>249</td>
</tr>
<tr>
<td>ViewAs</td>
<td>249</td>
</tr>
<tr>
<td>Constants</td>
<td>249</td>
</tr>
<tr>
<td>TXtnbHotSpotViewAs</td>
<td>249</td>
</tr>
<tr>
<td>TNEBHotSpots Component</td>
<td>250</td>
</tr>
<tr>
<td>Properties</td>
<td>250</td>
</tr>
<tr>
<td>BinData</td>
<td>250</td>
</tr>
<tr>
<td>Count</td>
<td>250</td>
</tr>
<tr>
<td>EndCol</td>
<td>251</td>
</tr>
<tr>
<td>EndRow</td>
<td>251</td>
</tr>
<tr>
<td>HotSpotName</td>
<td>251</td>
</tr>
<tr>
<td>Index</td>
<td>252</td>
</tr>
<tr>
<td>Items</td>
<td>252</td>
</tr>
<tr>
<td>ProfilesControl</td>
<td>253</td>
</tr>
<tr>
<td>StartCol</td>
<td>253</td>
</tr>
<tr>
<td>StartRow</td>
<td>253</td>
</tr>
<tr>
<td>Subkey</td>
<td>254</td>
</tr>
<tr>
<td>ViewAs</td>
<td>254</td>
</tr>
<tr>
<td>Methods</td>
<td>254</td>
</tr>
<tr>
<td>AboutBox</td>
<td>254</td>
</tr>
<tr>
<td>Add</td>
<td>255</td>
</tr>
<tr>
<td>AsVariant</td>
<td>255</td>
</tr>
<tr>
<td>Delete</td>
<td>255</td>
</tr>
<tr>
<td>LoadFromXMLFile</td>
<td>255</td>
</tr>
<tr>
<td>SaveToXMLFile</td>
<td>256</td>
</tr>
<tr>
<td>ShowToXMLFile</td>
<td>256</td>
</tr>
<tr>
<td>Helper Controls</td>
<td>256</td>
</tr>
<tr>
<td>TNEBSync Control</td>
<td>257</td>
</tr>
<tr>
<td>TNEBSync Control</td>
<td>257</td>
</tr>
<tr>
<td>Properties</td>
<td>257</td>
</tr>
<tr>
<td>EmptyScreenDelay</td>
<td>257</td>
</tr>
<tr>
<td>TelnetControl</td>
<td>258</td>
</tr>
<tr>
<td>TraceControl</td>
<td>258</td>
</tr>
<tr>
<td>WaitControl</td>
<td>258</td>
</tr>
<tr>
<td>Methods</td>
<td>259</td>
</tr>
<tr>
<td>AboutBox</td>
<td>259</td>
</tr>
<tr>
<td>AsVariant</td>
<td>259</td>
</tr>
<tr>
<td>Connect</td>
<td>260</td>
</tr>
<tr>
<td>Disconnect</td>
<td>260</td>
</tr>
<tr>
<td>IsConnected</td>
<td>260</td>
</tr>
<tr>
<td>SendKeys</td>
<td>261</td>
</tr>
<tr>
<td>Wait</td>
<td>262</td>
</tr>
<tr>
<td>WaitFor</td>
<td>263</td>
</tr>
<tr>
<td>WaitForConnect</td>
<td>263</td>
</tr>
<tr>
<td>WaitForData</td>
<td>264</td>
</tr>
</tbody>
</table>

© 2015, Cybele Software, Inc.
<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitForDisconnect</td>
<td>264</td>
</tr>
<tr>
<td>WaitForEndDoc</td>
<td>265</td>
</tr>
<tr>
<td>WaitForNewScreen</td>
<td>265</td>
</tr>
<tr>
<td>WaitForNewUnlock</td>
<td>266</td>
</tr>
<tr>
<td>WaitForScreen</td>
<td>266</td>
</tr>
<tr>
<td>WaitForStartDoc</td>
<td>267</td>
</tr>
<tr>
<td>WaitForUnlock</td>
<td>267</td>
</tr>
<tr>
<td>TTNBXXmlTemplate Class</td>
<td>268</td>
</tr>
<tr>
<td>Properties</td>
<td>268</td>
</tr>
<tr>
<td>id</td>
<td>268</td>
</tr>
<tr>
<td>Methods</td>
<td>269</td>
</tr>
<tr>
<td>CanProduce</td>
<td>269</td>
</tr>
<tr>
<td>HostToXml</td>
<td>269</td>
</tr>
<tr>
<td>XmlToHost</td>
<td>269</td>
</tr>
<tr>
<td>Events</td>
<td>270</td>
</tr>
<tr>
<td>OnCustomXml</td>
<td>270</td>
</tr>
<tr>
<td>TNBXXmlBroker Reference</td>
<td>270</td>
</tr>
<tr>
<td>TNBXXmlBroker Control</td>
<td>270</td>
</tr>
<tr>
<td>Properties</td>
<td>270</td>
</tr>
<tr>
<td>IncludeStatus</td>
<td>270</td>
</tr>
<tr>
<td>Producer</td>
<td>271</td>
</tr>
<tr>
<td>ScreenName</td>
<td>271</td>
</tr>
<tr>
<td>TelnetControl</td>
<td>271</td>
</tr>
<tr>
<td>XML</td>
<td>271</td>
</tr>
<tr>
<td>Methods</td>
<td>272</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>272</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>272</td>
</tr>
<tr>
<td>XML Reference</td>
<td>272</td>
</tr>
<tr>
<td>XML Tags and Attributes</td>
<td>273</td>
</tr>
<tr>
<td>TNBRIDGE</td>
<td>273</td>
</tr>
<tr>
<td>Status</td>
<td>273</td>
</tr>
<tr>
<td>Screen</td>
<td>273</td>
</tr>
<tr>
<td>XML Example</td>
<td>274</td>
</tr>
<tr>
<td>TNBXXmlClient Reference</td>
<td>274</td>
</tr>
<tr>
<td>TNBXXmlClient Control</td>
<td>274</td>
</tr>
<tr>
<td>Properties</td>
<td>275</td>
</tr>
<tr>
<td>AidKey</td>
<td>275</td>
</tr>
<tr>
<td>AidString</td>
<td>277</td>
</tr>
<tr>
<td>ExcludeEmptyFields</td>
<td>278</td>
</tr>
<tr>
<td>HotSpotsControl</td>
<td>278</td>
</tr>
<tr>
<td>ScreenRow</td>
<td>278</td>
</tr>
<tr>
<td>Text</td>
<td>278</td>
</tr>
<tr>
<td>XML</td>
<td>279</td>
</tr>
<tr>
<td>Methods</td>
<td>279</td>
</tr>
<tr>
<td>Connect</td>
<td>279</td>
</tr>
<tr>
<td>Disconnect</td>
<td>279</td>
</tr>
<tr>
<td>GetScreenRowEx</td>
<td>280</td>
</tr>
<tr>
<td>GetScreenText</td>
<td>280</td>
</tr>
<tr>
<td>IsConnected</td>
<td>281</td>
</tr>
<tr>
<td>FindEditField</td>
<td>281</td>
</tr>
<tr>
<td>LoadFromXMLFile</td>
<td>282</td>
</tr>
<tr>
<td>SaveToXMLFile</td>
<td>282</td>
</tr>
<tr>
<td>SendAid</td>
<td>282</td>
</tr>
</tbody>
</table>
## TN Bridge Trace Services

### Setting up Trace Services

### TN Bridge Trace Server

### TNBXTrace Control

#### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td>300</td>
</tr>
<tr>
<td>Directory</td>
<td>300</td>
</tr>
<tr>
<td>HidePasswordFields</td>
<td>301</td>
</tr>
<tr>
<td>Port</td>
<td>301</td>
</tr>
<tr>
<td>PurgeTraceFiles</td>
<td>301</td>
</tr>
<tr>
<td>TelnetControl</td>
<td>302</td>
</tr>
</tbody>
</table>
2 Delphi Components .......................................................................................................................................................... 306

Syntax conventions ............................................................................................................................................................ 306

Mainframe Access Components ..................................................................................................................................... 307

TTnb5250/TTnb3270 Reference ......................................................................................................................................... 307

TTnb5250/TTnb3270 Components ...................................................................................................................................... 307

Properties .............................................................................................................................................................................. 308

- AidKey ............................................................................................................................................................................. 308
- CodePage .......................................................................................................................................................................... 309
- Cols .................................................................................................................................................................................... 310
- ConnectionName .............................................................................................................................................................. 311
- ConversionTable ............................................................................................................................................................... 311
- Debug ................................................................................................................................................................................ 312
- DebugDir ............................................................................................................................................................................ 313
- DeviceName ..................................................................................................................................................................... 313
- EditFields ......................................................................................................................................................................... 314
- ExcludeEmptyFields ......................................................................................................................................................... 314
- Extended ........................................................................................................................................................................... 314
- Host .................................................................................................................................................................................. 315
- HostFields ....................................................................................................................................................................... 315
- IsExtended ......................................................................................................................................................................... 316
- KeepAlive .......................................................................................................................................................................... 316
- LastErrMsg ....................................................................................................................................................................... 317
- Password ........................................................................................................................................................................... 317
- Port .................................................................................................................................................................................. 317
- Rows ................................................................................................................................................................................ 318
- ScreenRow ....................................................................................................................................................................... 318
- SessionState ................................................................................................................................................................... 319
- SubKey .............................................................................................................................................................................. 320
- TerminalType ................................................................................................................................................................. 320
- TnbProfiles ..................................................................................................................................................................... 321
- TnbTrace .......................................................................................................................................................................... 322
- UserId ............................................................................................................................................................................... 322
- XLockDelay .................................................................................................................................................................... 322

Methods .................................................................................................................................................................................... 323

- About ................................................................................................................................................................................ 323
- ClassNames ................................................................................................................................................................. 323
- CloseEditor ................................................................................................................................................................. 323
- Connect ............................................................................................................................................................................ 324
- Disconnect ................................................................................................................................................................. 324
- FromPool ....................................................................................................................................................................... 325
- GetScreenRowEx .......................................................................................................................................................... 325
- GetScreenText ............................................................................................................................................................ 326
- IsConnected ................................................................................................................................................................. 327
- LoadFromFile ............................................................................................................................................................... 327
- Release ........................................................................................................................................................................... 327
- SaveToFile ..................................................................................................................................................................... 328
SendAid ................................................................. 328
SendKeys .............................................................. 329
Show Editor ......................................................... 330
Events ................................................................. 331
  OnConnect ......................................................... 331
  OnConnectionFail ............................................... 331
  OnDisconnect .................................................... 331
  OnScreenChange ............................................... 332
  OnSendAid ......................................................... 332
  OnSessionState ................................................ 332
  OnSystemLock .................................................. 332
  OnSystemUnlock ............................................... 333
TTnb3287 Reference .............................................. 333
TTnb3287 Component .......................................... 333
  Properties ......................................................... 334
  AttachMode ....................................................... 334
  CodePage .......................................................... 334
  ConversionTable ............................................... 335
  Debug ............................................................... 336
  DebugDir .......................................................... 336
  DeviceName ...................................................... 337
  Host ................................................................. 337
  IsConnected ..................................................... 338
  Port ................................................................. 338
  Printer .............................................................. 339
  SendToPrinter ................................................... 339
  SubKey ............................................................. 339
  Text ................................................................. 340
  TnbProfiles ...................................................... 340
  Methods ........................................................... 340
  Connect ............................................................ 340
  Disconnect ......................................................... 341
  Show PrinterEditor ............................................. 341
  Events ............................................................ 342
  OnConnect ......................................................... 342
  OnConnectionFail ............................................. 342
  OnDataAvailable ............................................... 342
  OnDisconnect .................................................... 342
  OnEndDocument ................................................ 342
  OnStartDocument ............................................... 342
TTnb3812 Reference .............................................. 343
TTnb3812 Component .......................................... 343
  Properties ......................................................... 343
  CodePage .......................................................... 343
  ConversionTable ............................................... 344
  Debug ............................................................... 345
  DebugDir .......................................................... 346
  DeviceName ...................................................... 346
  Host ................................................................. 346
  IsConnected ..................................................... 347
  MsgQueueLib .................................................... 347
  MsgQueueName ................................................ 348
  Port ................................................................. 348
  Printer .............................................................. 348

© 2015, Cybele Software, Inc.
| AbstratString                       | 368 |
| Append                              | 368 |
| Ascii                               | 369 |
| BlkSize                             | 369 |
| BufSize                             | 370 |
| Bytes                               | 370 |
| CrLf                                | 370 |
| Direction                           | 371 |
| FICommand                           | 371 |
| FIMode                              | 372 |
| FIMName                             | 372 |
| HostFileName                        | 372 |
| HostKind                            | 373 |
| LocalFileName                       | 374 |
| LRecI                               | 374 |
| PrimSpace                           | 375 |
| RecFm                               | 375 |
| SecSpace                            | 376 |
| ShowDialog                          | 376 |
| ShowEditor                          | 376 |
| Stream                              | 377 |
| SubKey                              | 378 |
| Timeout                             | 378 |
| TnbCom                              | 378 |
| TnbProfiles                          | 379 |
| Units                               | 379 |
| Methods                             | 380 |
| Abort                               | 380 |
| LoadFromXMLFile                     | 380 |
| Receive                             | 380 |
| Run                                 | 381 |
| SaveToXMLFile                       | 381 |
| Send                                | 382 |
| Events                              | 382 |
| OnAborting                          | 382 |
| OnComplete                          | 382 |
| OnRunning                           | 383 |
| OnUpdateLength                      | 383 |
| Constants                           | 384 |
| FTM_Mode                            | 384 |
| THostKind                           | 384 |
| Constants                           | 384 |
| TNBCP                               | 384 |
| TNBAid                              | 385 |
| TNBSS                               | 386 |
| TNB5250TT                           | 386 |
| TNB3270TT                           | 386 |
| Terminal Emulator Components        | 387 |
| TTNbEmulator Component              | 387 |
| Properties                          | 387 |
| CBId                                | 388 |
| CursorPos                           | 388 |
| DisableKbdLockOnError               | 389 |
TTnbHotSpots Class Reference ................................................................. 431
TTnbHotSpots Class Reference ................................................................. 431
Properties ..................................................................................... 432
  ActionFieldData ................................................................. 432
  Active ................................................................................. 433
  Background .................................................................... 433
  Color ............................................................................ 434
  Description .................................................................. 434
  EndCol .......................................................................... 434
  EndRow .......................................................................... 435
  Id .................................................................................. 435
  Pattern ........................................................................... 436
  StartCol .......................................................................... 437
  StartRow ......................................................................... 437
  TextColor ....................................................................... 438
  ViewAs ........................................................................... 438
Constants ................................................................................... 439
  TTnbHotSpotViewAs .................................................. 439
TTnbHotSpots Component ................................................................ 439
TTnbHotSpots Component ................................................................ 439
Properties ................................................................................... 440
  EndCol .......................................................................... 440
  EndRow .......................................................................... 440
  HotSpotName ................................................................ 441
  Items ............................................................................. 441
  StartCol .......................................................................... 442
  StartRow ......................................................................... 442
  Subkey ........................................................................... 443
  TnbProfiles .................................................................... 443
  ViewAs ........................................................................... 443
Methods ...................................................................................... 444
  About ............................................................................ 444
  Add ............................................................................... 444
  Delete ............................................................................ 445
  LoadFromFile .................................................................. 445
  SaveToFile ...................................................................... 445
  Show Editor ..................................................................... 446
Helper Controls ............................................................................. 446
TTnbSync Component ..................................................................... 446
TTnbSync Component ..................................................................... 446
Properties ...................................................................................... 447
  EmptyScreenDelay ................................................... 447
  TnbCom .......................................................................... 447
  TnbTrace ......................................................................... 448
  WaitTimeout ................................................................. 448
Methods ...................................................................................... 449
  Connect .......................................................................... 449
  Disconnect ....................................................................... 449
<table>
<thead>
<tr>
<th>Method/Property</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsConnected</td>
<td>450</td>
</tr>
<tr>
<td>SendKeys</td>
<td>450</td>
</tr>
<tr>
<td>Wait</td>
<td>452</td>
</tr>
<tr>
<td>WaitFor</td>
<td>452</td>
</tr>
<tr>
<td>WaitForConnect</td>
<td>453</td>
</tr>
<tr>
<td>WaitForEndDoc</td>
<td>453</td>
</tr>
<tr>
<td>WaitForData</td>
<td>454</td>
</tr>
<tr>
<td>WaitForDisconnect</td>
<td>455</td>
</tr>
<tr>
<td>WaitForNewScreen</td>
<td>455</td>
</tr>
<tr>
<td>WaitForNewUnlock</td>
<td>456</td>
</tr>
<tr>
<td>WaitForStartDoc</td>
<td>456</td>
</tr>
<tr>
<td>WaitForScreen</td>
<td>457</td>
</tr>
<tr>
<td>WaitForUnlock</td>
<td>458</td>
</tr>
<tr>
<td>TTnbXmlTemplate Class</td>
<td>458</td>
</tr>
<tr>
<td>TTnbXmlTemplate Class</td>
<td>458</td>
</tr>
<tr>
<td>Properties</td>
<td>459</td>
</tr>
<tr>
<td>Id</td>
<td>459</td>
</tr>
<tr>
<td>Methods</td>
<td>459</td>
</tr>
<tr>
<td>CanProduce</td>
<td>459</td>
</tr>
<tr>
<td>HostToXml</td>
<td>460</td>
</tr>
<tr>
<td>XmlToHost</td>
<td>460</td>
</tr>
<tr>
<td>Events</td>
<td>461</td>
</tr>
<tr>
<td>OnCustomXml</td>
<td>461</td>
</tr>
<tr>
<td>TTnbXmlBroker Component</td>
<td>461</td>
</tr>
<tr>
<td>TTnbXmlBroker Component</td>
<td>461</td>
</tr>
<tr>
<td>Properties</td>
<td>461</td>
</tr>
<tr>
<td>IncludeStatus</td>
<td>461</td>
</tr>
<tr>
<td>Producer</td>
<td>462</td>
</tr>
<tr>
<td>ScreenName</td>
<td>462</td>
</tr>
<tr>
<td>TnbCom</td>
<td>462</td>
</tr>
<tr>
<td>XML</td>
<td>463</td>
</tr>
<tr>
<td>Methods</td>
<td>463</td>
</tr>
<tr>
<td>LoadFromFile</td>
<td>463</td>
</tr>
<tr>
<td>SaveToFile</td>
<td>463</td>
</tr>
<tr>
<td>XML Reference</td>
<td>464</td>
</tr>
<tr>
<td>XML Tags and Attributes</td>
<td>465</td>
</tr>
<tr>
<td>TNBRIDGE</td>
<td>465</td>
</tr>
<tr>
<td>Status</td>
<td>465</td>
</tr>
<tr>
<td>Screen</td>
<td>465</td>
</tr>
<tr>
<td>XML Example</td>
<td>465</td>
</tr>
<tr>
<td>TTnbXmlClient Component</td>
<td>466</td>
</tr>
<tr>
<td>TTnbXmlClient Component</td>
<td>466</td>
</tr>
<tr>
<td>Properties</td>
<td>467</td>
</tr>
<tr>
<td>AidKey</td>
<td>467</td>
</tr>
<tr>
<td>AidString</td>
<td>469</td>
</tr>
<tr>
<td>ExcludeEmptyFields</td>
<td>470</td>
</tr>
<tr>
<td>ScreenRow</td>
<td>470</td>
</tr>
<tr>
<td>Text</td>
<td>470</td>
</tr>
<tr>
<td>TnbHotSpots</td>
<td>471</td>
</tr>
<tr>
<td>XML</td>
<td>471</td>
</tr>
<tr>
<td>Methods</td>
<td>472</td>
</tr>
<tr>
<td>Connect</td>
<td>472</td>
</tr>
<tr>
<td>Disconnect</td>
<td>472</td>
</tr>
<tr>
<td>GetScreenRow</td>
<td>472</td>
</tr>
</tbody>
</table>
# OHIO Reference

## 1 Introduction

- Ohio containment hierarchy .................................................. 501
- Ohio inheritance hierarchy .................................................. 503

## 2 Programming Reference

### 2.1 Ohio Base Class

- Syntax Conventions ............................................................... 504
- Methods ............................................................... 504
- CreateOhioPosition ......................................................... 504
- Properties ................................................................. 505
- OhioVersion ............................................................... 505
- VendorName ................................................................. 505
- VendorObject ............................................................... 506
- VendorProductVersion ..................................................... 506

### 2.2 Class Definition

- Syntax Conventions ............................................................... 507
- OhioManager ................................................................. 507
- Methods ............................................................... 508
  - OpenSession ................................................................. 508
  - CloseSession ................................................................. 509
- Properties ................................................................. 509
- Sessions ................................................................. 510
- OhioSessions ................................................................. 510
- Methods ............................................................... 510
  - AddSession ................................................................. 510
  - CloseSession ................................................................. 511
- Properties ................................................................. 512
  - Count ................................................................. 512
  - Item ................................................................. 512
  - Manager ................................................................. 513
- OhioSession ................................................................. 513
- Methods ............................................................... 513
  - Connect ................................................................. 513
  - Disconnect ................................................................. 514
  - WaitForConnect ................................................................. 514
  - WaitForDisconnect ................................................................. 515
- Properties ................................................................. 515
  - ConfigurationResource ................................................. 516
  - Connected ................................................................. 516
  - Manager ................................................................. 516

---

© 2015, Cybele Software, Inc.
| Screen | .......................................................... | 517 |
| SessionName | .......................................................... | 517 |
| SessionState | .......................................................... | 517 |
| SessionType | .......................................................... | 518 |
| Trace | .......................................................... | 518 |
| Events | .......................................................... | 518 |
| OnConnect | .......................................................... | 519 |
| OnConnectionFail | .......................................................... | 519 |
| OnDisconnect | .......................................................... | 519 |
| OnScreenChange | .......................................................... | 519 |
| OnSendAid | .......................................................... | 519 |
| OnSessionState | .......................................................... | 520 |
| OhioScreen | .......................................................... | 520 |
| Methods | .......................................................... | 520 |
| GetData | .......................................................... | 521 |
| FindString | .......................................................... | 521 |
| Refresh | .......................................................... | 522 |
| SendAid | .......................................................... | 523 |
| SendKeys | .......................................................... | 523 |
| SetString | .......................................................... | 524 |
| Wait | .......................................................... | 524 |
| WaitFor | .......................................................... | 525 |
| WaitForNewScreen | .......................................................... | 525 |
| WaitForNewUnlock | .......................................................... | 526 |
| WaitForScreen | .......................................................... | 526 |
| WaitForUnlock | .......................................................... | 527 |
| Properties | .......................................................... | 527 |
| Columns | .......................................................... | 528 |
| Cursor | .......................................................... | 528 |
| Fields | .......................................................... | 529 |
| OIA | .......................................................... | 529 |
| Rows | .......................................................... | 530 |
| String/Text | .......................................................... | 530 |
| Events | .......................................................... | 530 |
| OnConnect | .......................................................... | 531 |
| OnConnectionFail | .......................................................... | 531 |
| OnDisconnect | .......................................................... | 531 |
| OnScreenChange | .......................................................... | 531 |
| OnSendAid | .......................................................... | 531 |
| OnSessionState | .......................................................... | 532 |
| OhioFields | .......................................................... | 532 |
| Methods | .......................................................... | 532 |
| Item | .......................................................... | 532 |
| FindByPosition | .......................................................... | 533 |
| FindByString | .......................................................... | 533 |
| Refresh | .......................................................... | 534 |
| Properties | .......................................................... | 535 |
| Columns | .......................................................... | 535 |
| Count | .......................................................... | 535 |
| Rows | .......................................................... | 536 |
| OhioField | .......................................................... | 536 |
| Methods | .......................................................... | 536 |
| GetData | .......................................................... | 536 |
| Properties | .......................................................... | 537 |
XML Reference

1 Introduction ........................................................................................................ 553

2 Programming Reference ...................................................................................... 554

   XML Tags and Attributes ...................................................................................... 555
      TNBRIDGE ........................................................................................................ 555
      Connections ..................................................................................................... 555
      ScreenStyles .................................................................................................... 556
      KeyboardMaps .................................................................................................. 557
      HotSpots .......................................................................................................... 558
      TnbMacros ........................................................................................................ 559

   XML Example .................................................................................................... 560

Purchasing TN Bridge Host Integration Pack ......................................................... 561

1 Licensing Information .......................................................................................... 562

2 How to Place an Order ......................................................................................... 562

3 How to register your License ............................................................................... 563
      ActiveX Edition ................................................................................................ 563
      Delphi Edition ................................................................................................ 565

4 Obtaining Technical Support .............................................................................. 566

© 2015, Cybele Software, Inc.
1 Introduction

1.1 Overview

**TN BRIDGE Host Integration Pack 3.5** is a set of components and productivity tools to extend and take advantage of already developed and tested screen-oriented host applications. It provides full connectivity to IBM S/390 and AS/400 systems through standard telnet protocol.

Offering a set of 100% ActiveX and Borland’s Delphi-native components (.NET managed code also available), it allows development under the most popular development environments and programming languages like Microsoft® Visual Basic, VB.NET, C#, ASP, ASP.NET and Borland’s Delphi 6 thru Delphi 2005.

Extending legacy applications with Host Integration Pack is much more profitable than starting all over again. It does not requires any kind of extra code in your host, allowing reuse of the already tested and developed host applications.

With Host Integration Pack you won’t need to learn about complex communication protocols nor low level API’s. Projects development-time are measured in days, not months. Your end-users will obtain new applications in a familiar and more friendly environment, reducing training time and increasing their’s productivity.

Host Integration Pack offers a simplified non-event-driven programming model, making most of developments much easier. However, it still allows low-level access and event-driven programming, more suitable for developing complex applications.

Includes a productivity environment that helps during the design and development-cycle of host-integrated applications. This tool drastically reduces the training curve and testing time.

Through OHIO Interface, now an integrated feature of TN BRIDGE Host Integration Pack, you can run an OHIO application without having an emulation session running, you can concentrate on the application functions, without worrying about structure packing details or parameter command codes. OHIO also has all the benefits of the object oriented programming paradigm because it was designed as an object oriented API.

Host Integration Pack offers a powerful XML interface to the host, enabling a fast path to integrate with XML Web Services, applications in Microsoft’s .Net architecture, J2EE applications, or interact with robust applications including IBM’s WebSphere and Microsoft’s BizTalk Servers.
1.2 Architecture

TN BRIDGE Host Integration Pack contains remote printing and file transfer capabilities as well as a productivity environment. Development Lab is the research tool for TN BRIDGE Host Integration Pack, tracking and testing environment, which reduces the training curve, as well as development and testing times, while ensures a more reliable application.

1.3 What's New in 3.5?

Following are the new implementations in Host Integration Pack 3.5:

- New XML-To-Host bridging classes.
- Improved Development Lab with XML templates builder.
Following are the new implementations in Host Integration Pack 3.0:

- **Extended OHIO Interface.** By implementing this internationally accepted standard API, Host Integration Pack enables corporations to protect their investment, as well as an easy and smooth migration path to the superior TN BRIDGE technology. TN BRIDGE Extensions to standard OHIO allows taking full advantage of the Host Integration Pack synchronous (non-event-driven) model, as well as access to other Host Integration Pack's exclusive features, like XML streaming interfaces.

- **New XML Broker and XML Client components.** Host Integration Pack offers a powerful XML interface, enabling a fast path to integrate with XML Web Services, applications in Microsoft's .Net architecture, J2EE applications, or interact with robust applications including IBM's WebSphere and Microsoft's BizTalk Servers.

- **New XML Configuration for Profiles component.** In this version, each TnbProfile component can be save in configurations in both OLE Compound Documents and XML formats.

- **LPD Component.** Host Integration Pack 3.0 now includes a Line Printer Deamon Component with SCS (SNA Character String) interface, emulating IBM 3287 and 3812 printers.

### 1.4 Trial & Full License

All our products have a 30 days free trial period. We offer full pre-sales support. Do not hesitate to contact us if you require assistance or clarification to develop your project.

**Evaluation Period**

If you downloaded **TN Bridge Host Integration Pack**'s Trial Version from our website or a distribution site, our software will work in Demo mode, allowing time-limited connections to the mainframe.

Please get registered to activate the full demo mode during the 30 days free trial period:
By filling a short form you will be automatically emailed a trial key file that you must save into the Windows directory and in the folder where your TN Bridge is installed:

For example: C:\Program Files\TN BRIDGE Host Integration Pack 3.5

In case you need to extend your evaluation period, please contact us. We will gladly provide you with a trial extension key.

Registering your Full Edition

To register your purchased TN Bridge license, read the following topic:
- How to Register your License.

2 The Development Lab

TN BRIDGE Development Lab is a research, tracking and testing environment which integrates tools for easy development of TN BRIDGE-based applications. Development Lab reduce the learning curve and testing time, thus, development time is much less shorter. This unified environment involves a terminal emulator, trace viewer and interactive scripting, giving the chance to easily understand host events, inspect screens and fields, interactively test TN BRIDGE components, create code snippets, and more.

TN BRIDGE Development Lab will help you develop your applications in these scenarios:

- During planning/design time, because you will understand the screen/events sequences produced by your host application. Then you can plan and test your navigation strategy through the host screens. You can do this working in Interactive mode.

- During development time, using the remote tracking functionality where you can connect to your TN BRIDGE-based development, even running on another computer. You can do this working in On-Line Trace Mode.

To keep your deployed application free of errors. You can work in On-Line Trace Mode in order to connect and track a running application even in a remote workstation or locally save the trace information into a file for posterior analysis working in Off-Line Trace Mode.

You can also save screens as XML files and connect to them to work with mainframe screens being off-line.
2.1 Development Lab components

TN BRIDGE Development Lab's main window contains several panels offering access to the different functionalities.

- **Main menu and toolbars**: Application settings and actions can be accessed through menu items and buttons.
- **The Display workspace**: Provides access to the interactive emulation, screen snapshot, XML snapshot and interactive scripting editor.
- **The Trace Event views**: Provides tree and list views to TN BRIDGE's communication events.
- **The Object Inspector**: Displays information about screen-fields and XML objects.
2.1.1 Main Menu and Toolbar

You can access to functions through the main menu items and toolbar buttons.

- **Main Menu** offers four submenus for accessing to actions and settings.
- **Main Toolbar** The toolbar buttons offer quick access to the most commonly used functions.

2.1.1.1 Main Menu

At the Main Menu you can find the following options:
File

Open
Opens a TN BRIDGE Trace File.

Select XML Template
Allows you to send a hard-copy of the screen to the printer.

Change XML Template Folder
Use this option to display the standard printer setup dialog.

Save Screen As
Save the current screen in XML format.

Print
Allows you to send a hard-copy of the screen to the printer.

Exit
Terminates z/Scope Classic.

Edit

Copy
Use this option to copy the selected text-area into clipboard buffer.

Paste
Use this option to paste text from the clipboard into the screen at the cursor position.
View

<table>
<thead>
<tr>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen</td>
</tr>
<tr>
<td>Event Tree</td>
</tr>
<tr>
<td>Event List</td>
</tr>
<tr>
<td>Object Inspector</td>
</tr>
</tbody>
</table>

**Style**
Allows you to choose a different Application Skin.

**Screen**
Shows/Hides the display window

**Event Tree**
Shows/Hides the Event Tree window

**Event List**
Shows/Hides the Event List window

**Object Inspector**
Shows/Hides the Object Inspector window

Help

<table>
<thead>
<tr>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>TN BRIDGE on the Web</td>
</tr>
<tr>
<td>Support</td>
</tr>
<tr>
<td>About</td>
</tr>
</tbody>
</table>

**Help**
Use this option to get access to this help.

**TN BRIDGE on the Web**
Visit our Web Site.

**Support**
Send us your feedback

**About**
Shows you information about TN BRIDGE Development Lab.
2.1.1.2 Main Toolbar

**Connections**

- **Connect/Disconnect**
  Use this button to connect and disconnect the current connection. This connection's name is displayed at selected session tab.

- **Open**
  Opens a TN BRIDGE Trace File

- **Save Screen As**
  Save the current screen in XML format.

**Edit**

- **Copy text**
  Use this button to copy a selected text-area into clipboard buffer.

- **Paste text**
  Use this button to paste text from the clipboard into the screen at the cursor position.

**Preferences**

- **Settings**
  Use this button to access to Development Lab Preferences window.

**Print**

- **Print screen**
  Use this button to send a hard-copy of the screen to the printer.

**Help**
The display workspace allows access to the following windows:

- **Interactive emulation**
- **Screen snapshot**
- **XML views**
- **Scripting window**

2.1.2.1 Interactive emulation

The interactive emulation window allows you to get connected to your mainframe and take the mainframe connection as source of tracing windows, XML transforming and interactive scripting.
The statusbar brings you information about connection status, system status and input status. Additionally, it shows application messages. The status bar is divided into five panels as described below:

- Connection status.
- System/Input status.
- Insert/Overwrite mode.
- Screen-cursor coordinates.
- Messages area.

See also **Interactive** mode, which explains how you can create a connection setting and get connected to your mainframe.

### 2.1.2.2 Screen Snapshot

The screen snapshot window shows a snapshot of a mainframe screen. In interactive mode, you can switch to snapshot window by clicking on "Snapshot" tab or selecting a screen in any of the trace view windows.

See also **Trace Event views**
2.1.2.3 XML views

The XML view windows allows to see a XML representation of the current screen. The XML representation can be generic or the one resulting by applying an XML template.

Two views are available:

- **XML Tree view:**

![XML Tree view]

- **XML Text view**

![XML Text view]

2.1.2.4 Scripting window

The scripting window shows the scripting editor, where you can create code snippets to test the mainframe connection.
2.1.3 The Trace Event views

The Trace window shows the events generated by TN BRIDGE in relation to the communication with the mainframe. This information can be gathered from the interactive emulation, from an external application using the TN BRIDGE components or from file.

Trace events are displayed in two views:

- **as Tree view**, where events are displayed in hierarchical way.
- **as List view**, where events are displayed as a list.

2.1.3.1 Tree View

Events are nested by connection and send event.
Connect method

Represent a Connect method call.

General Event

Represent the following events:
- OnConnect
- OnDisconnect
- OnSystemLock
- OnSystemUnlock
- OnSessionState
- OnConnectionFail

OnScreenChange event

Represent the OnScreenChange event and the associated screen data received.

SendAid method

Represent a SendAid method call (or any SendAid related methods or properties).

OnSendAid event

Represent the OnSendAid event and the associated screen data sent.

General Method

Represent a method call.

Script execution

Represent the execution of a script code block (an ASP page) using TN BRIDGE controls. All TN BRIDGE methods and events between the start and the end of the script will be included here.

The Trace Entries pane shows a tree with TN BRIDGE's method calls, communication
events and custom trace data. Each method call entry is shown as a group, nesting the upcoming data (including another method calls) until it ends.

2.1.3.2 List View

The Event List has a record of all the events processed during the session. It also displays additional information such as the exact time of the event reception and the elapsed time it took to be triggered.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time Stamp</th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>OnConnect</td>
<td>14:43:03,01</td>
<td>240 ms.</td>
</tr>
<tr>
<td>OnSessionStart</td>
<td>14:43:04,19</td>
<td>1182 ms.</td>
</tr>
<tr>
<td>OnSessionEnd</td>
<td>14:43:04,20</td>
<td>10 ms.</td>
</tr>
<tr>
<td>OnSessionEnd</td>
<td>14:43:04,20</td>
<td>0 ms.</td>
</tr>
<tr>
<td>OnScreenChange</td>
<td>14:43:04,20</td>
<td>0 ms.</td>
</tr>
<tr>
<td>OnSystemLock</td>
<td>14:43:04,82</td>
<td>621 ms.</td>
</tr>
<tr>
<td>OnScreenChange</td>
<td>14:43:04,84</td>
<td>0 ms.</td>
</tr>
<tr>
<td>OnSystemUnlock</td>
<td>14:43:05,02</td>
<td>180 ms.</td>
</tr>
</tbody>
</table>

2.1.4 The Object Inspector

The Object Inspector allows to inspect attributes of screen-fields and XML fields.

- To inspect the attributes of an specific field, double-click on it.
2.2 Working with TN BRIDGE Development Lab

In this chapter we'll see how to use the main functionalities provided by TN BRIDGE Development Lab.

- **Development Lab modes** details the modes of operation of Development Lab.
- **Scripting** explains how to create code snippets to test the host connection and play with TN BRIDGE objects.
- **Working with XML** explains how to use Development Lab to create XML templates.

2.2.1 Development Lab modes

Main use of TN BRIDGE Development Lab is tracing TN BRIDGE-based applications. It provides three modes of operation: interactive, on-line trace and off-line trace.

- **Interactive** explains how to use is mode in which we can access to a mainframe using a built-in terminal emulator and watch generated events and fields.
- **On-Line Tracing** explains how to connect and trace TN BRIDGE applications.
- **Off-Line Tracing** explains how to open and read trace files generated by TN BRIDGE applications.
2.2.1.1 Interactive

This mode provides a built-in terminal emulator which can be used to get connected to a Tn3270 or Tn5250 host. Additionally, it allows to use XML files to simulate mainframe screens.

- **Creating a connection** shows how to create and configure a client-to-host connection.
- **Connecting to the mainframe** explains how to connect to the host using the previously saved connection.
- **Using XML connections** explains how to create and get connected to host screens simulated by XML files.

---

### 2.2.1.1.1 Creating a connection

To define a new connection to a mainframe click on Settings/Connection/3270 or 5250 popup item.
The following dialog will appear:

![Connection Setting](image)

After filling in the appropriate fields you must save the connection by clicking "Save As" button.

**Parameters**

**Host Address**
Host name (DNS) or IP address.

**Host Port**
Host connection port. Default value to standard telnet port (23).

**Keep Alive**
Enables keep-alive mechanism, needed for some telnet servers.

**Dump**
Enables generation of telnet communication logs.

**Terminal Type**
Sets the terminal type.
TN Extended
Enables Extended Telnet protocol (TN3270E or TN5250E).

Terminal Name
Logical unit or device name to be used in this connection.

Table Name
Load the character conversion table from an .ebc file.

Code Page
Allows the selection of an internal character conversion table.

Use Default
Resets the Code Page parameter to its default setting.

Defined connections
Shows and allows to choose previously defined connections.

2.2.1.1.2 Connecting to the mainframe

Once you have defined one or more host connections you'll be able to get connected to one of these by clicking on the arrow of the connect button and selecting the connection.

See also Creating a connection

2.2.1.1.3 Using XML Connections

XML connections allows you work with mainframe screens in an off-line fashion. To work in this mode, you need to create the XML files corresponding to the mainframe application you want to work with.

- **Saving a screen as an XML file** shows how to save screens in files with XML format.
- **Connecting to a set of XML screen files** Loads a previously persisted screen from an XML file.
2.2.1.1.3.1 Saving screens as XML files

To save a screen in XML format you must do the following:

- Go to the mainframe screen you want to save

- Click on Save Screen button
Repeat the operation for all the screens you want to see off-line.

2.2.1.3.2 Connecting to a set of XML screen files

To connect to a set of XML-screen files, follow the steps below:

1. Go to XML Connection Preferences.

2. Select the starting XML Screen file name you want to work with and save the connection; in our case we have saved it with the name XMLNetsahre.
3. To connect, click on your defined XML connection.

4. If the screen is displayed, you are having a successful XML offline session.
5. You can use PgDown/PgUp to move across the XML-screen files saved in the same directory.

2.2.1.2 On-Line tracing

On-Line Trace connection was created to access external applications using TN BRIDGE components.

- **Connecting to a TN BRIDGE Application** Shows how to trace a TN BRIDGE application running in your computer or even on a computer within your LAN.
2.2.1.2.1 Connecting to a TN BRIDGE Application

To connect to a remote TN BRIDGE application, start TN BRIDGE Development Lab and follow these steps:

- Select Settings\Trace Connections item.

- Enter the required information and click on add to save the connection.

**Parameters**
Name
Descriptive name for the connection.

Host
IP address or DNS name of the computer where the external TN BRIDGE application is running.

Port
TCP port specified in TN BRIDGE's Trace component.

- You are now ready to connect to a remote TN BRIDGE Application

2.2.1.3 Off-Line tracing
TN BRIDGE Development Lab allows you open trace files.

- **Opening a Trace File** Loads a trace file

---

**Off-Line Trace mode** allows you to analyze trace information from a file.

**Trace File**
Saved from a previous Lab session or from a TN BRIDGE application through the **Trace Server**.

**The Trace Entries Pane**
Displays all communication events as they are fired.

**The Screen Snapshot**
With it you can watch the screens and its fields on Windows-to-Host interaction.
2.2.1.3.1 Opening a Trace File

To open a previously saved trace file, select Open from the File Menu.

2.2.2 Scripting

Development Lab's main use is understanding the event flow and mainframe application's logic. Scripting inside Development Lab allows you to write code to test the mainframe logic and event flow from TN BRIDGE programming point of view. Also it can help you create code snippets to be included in your application.

Code can be written in Microsoft VB Script language, accessing methods and properties of Tn3270/Tn5250 components. A global object "Telnet" is available, associated to the Tn3270 or Tn5250 currently in use in the interactive emulation.

As an example, you can write the following code access a mainframe application:
To run the script click on Play button on the common toolbar.

To load a saved click on Open button of the common toolbar.

2.2.3 Working with XML

XML provides the facility to define tags and the structural relationship between them. As a result, developers can create their own customized tags (the extensible part of the puzzle) in order to define, share, and validate information between computing systems and applications.

Host Integration Pack allows you to build an XML bridge between your application and the mainframe.

- **Generic XML** explains the generic XML that TN BRIDGE provides by mapping screen-fields.
- **XML Templates** explains how we can create XML Templates to provide customized XML tags and attributes.
2.2.3.1 Generic XML

By default, screen information is mapped to XML tags and attributes as described:

```
<TNBRIDGE>
  <status>
    <direction... /> 
    <transaction... /> 
    <cursor_pos... /> 
    <session... /> 
    <state... /> 
    <xmlscreen... /> 
    <cursor_field... /> 
    <timestamp... /> 
  </status>
  <screen>
    <rows... /> 
    <cols... /> 
    <type... /> 
    <model... /> 
    <fields>
      <field ...>
        <name... /> 
        <attr... /> 
        <value... /> 
      </field>
      ...
      ...
    </fields>
  </screen>
</TNBRIDGE>
```

TNBRIDGE is the root tag and includes the status and screen tags as children.

**status**: indicates the connection status
Contains the following tags:

- **direction**: indicates INPUT or OUTPUT direction. When the XML is generated by the XmlBroker control it is set with OUTPUT value.
- **transaction**: indicates transaction identifier.
- **cursor_pos**: indicates cursor location.
- **session**: LU-LU (in session with VTAM application), noSession (not in session) or SSCPLU (in session with VTAM).
- **state**: indicates LOCKED or UNLOCKED state.
- **xmlscreen**: indicates the screen name. You must set this property using the ScreenName property.
- **cursor_field**: indicates the field's name where the cursor is located.
- **timestamp**: indicates the date the file was created.

**Screen**: indicates screen's characteristics and contains a collection of screen fields. Contains the following tags:
- **rows**: indicates the number of rows of the screen.
- **cols**: indicates the number of columns of the screen.
- **type**: indicates terminal type (3270 or 5250).
- **model**: indicates terminal model.
- **fields**: contains a collection of fields.

Here you can see an example of XML Code:

```xml
<TNBRIDGE>
  <status>
    <direction>OUTPUT</direction>
    <transaction>00A8DB7000002</transaction>
    <cursor_pos>830</cursor_pos>
    <session>LU-LU</session>
    <state>UNLOCKED</state>
    <xmlscreen name="Session" />
    <cursor_field>R11C30</cursor_field>
    <timestamp>2453360.14494451</timestamp>
  </status>
  <screen>
    <rows>24</rows>
    <cols>80</cols>
    <type>TN3270</type>
    <model>2E</model>
    <fields>
      <field name="R01C02">
        <value maxlen="8" len="8" value="KLGLGON1" />
      </field>
      <field name="R01C11">
        <value maxlen="13" len="13">-------------</value>
      </field>
    </fields>
  </screen>
</TNBRIDGE>
```
2.2.3.2 XML Templates

What is an XML Template?

By default, Host Integration Pack provides a generic XML representation. Because this generic XML contains complete screen information it is useful to recreate a screens on the consumer side. However, when it comes to real business information, it doesn't provide a real advantage regarding to accessing screen data directly.

Using Development Lab you have the possibility to generate XML Templates containing the necessary information to create tags and attributes representing the screen information as you perceive it, rather than using protocol data. This result in a much more legible and understandable XML file.

In this chapter we will analyze:

- **Enabling Template Mode** explains how to activate XML Template's functionalities.
- **Building an XML Template** shows how to persist data fields to an XML Template file.

Example XML generated file

You have now generated a nice readable XML Template file.
2.2.3.2.1 Enabling Template Mode

In order to work with XML Template files, you must first enable the XML Template mode.

- Click on the icon as shown below.

A window will ask you to specify a search path. The application will look for matching XML Templates for the current screen in the specified folder.

Keep in mind that XML Template files for a specified connection must be saved on the same directory, so they can be automatically recognized as the screen changes.

- To change the folder path do the following:

2.2.3.2.2 Building XML Templates

Building XML Templates involves recognizing and telling Development Lab where the data is located as well as its attributes. In some way, we need to define what data should be considered variable and what fixed text. This can be done both automatically and manually.

Automatic recognizing is the best option to start defining a XML Template, because it will do the larger and tedious work. However it's far beyond perfect, so you will need to make manual changes to attributes and sometimes manually redefine XML template data.

A XML Template is composed by XML Template items. Each XML Template item has an screen-area assigned as well as attributes that specifies how this item should behave at runtime.

Currently, XML Template items are divided into XML Template Fields, XML Template Tables and custom XML Template Areas.

- Generating XML Fields shows you how to automatically create XML template fields.
- Using the Object Inspector shows you how use the object inspector to modify
attributes of XML fields.

- **Identifying the screen** explain the importance of some fields in the XML template.
- **Deleting XML Template Fields** shows you how to delete XML template fields.
- **Saving an XML Template** explain how to save the XML template.
- **Adding XML Template Fields** shows you how to add a single XML Template field.
- **Adding XML Template Tables** shows you how to define a table.
- **Add custom XML Template Areas** shows you how create user-defined areas to populate at runtime.

### 2.2.3.2.2.1 Generating XML Template Fields

Using this option you can automatically generate XML-Template Fields inside of a selected area. An XML Template Field is an area that can have the following attributes:

- Label
- Text
- Label & Text

Text fields are those areas considered as they contain variable information and doesn't carry by themselves information about what type of information they contain. This fields have Label fields associated.

Label fields are considered those that gives information about what a Text field means.

Label & Text are considered those that carries both inherent information of the data if contains and variable data.

For example, in:

```
UserId: jdoe01
```

"UserId" is considered as a Label field an "jdoe01" as Text field, whose Label field is "UserId"

In image below, "User Taks", which is the title of the page, is considered Label & Text field, with label "Title".

In order to generate XML Template Fields select an area which contains the fields you want to add to the XML Template.

The resulting template fields are shown:
- Label fields enclosed in green brackets
- Text fields enclosed in red brackets
- Label & Text fields enclosed in white brackets
Note: You can now use the **Object Inspector** to customize the field you need.

Switching to XML view we get:
2.2.3.2.2.2 Using Object Inspector

The Object Inspector allows you to customize the fields which are going to be inserted in the XML Template.
Parameters

Identity
Determines if the field will be verified to match the current screen in order to assign the correct XML Template.

Note: The XML Template will be assigned to the screen only if all the identities with true value match.

Type
Defines the control's type.

Tag
Defines the XML tag the field will create.

2.2.3.2.2.3 Identifying the screen

Once you have created and saved a couple of XML Template files you will notice that when you retrieve from host a screen that already has a XML Template that matches, it is automatically selected and shown in the display. When a screen arrives, the XML Template engine looks in the template's directory for the template that own those XML Template Fields that have the "Identity" attribute set to true and match in the current screen.

When generating XML Template Fields in automatic way, these fields are in general determined by Development Lab. However as we said, this is far beyond perfect, so you might need to modify this attribute on some fields. Always take into account that "Identity" fields must be those that are unique to the screen and wont suffer any change neither in its content nor its location.

2.2.3.2.2.4 Deleting Template Fields

XML Template Fields can be deleted in two ways:

- right clicking on the field and selecting Delete XmlField
- selecting an screen area, right clicking and selecting Delete XmlFields.
2.2.3.2.2.5 Saving an XML Template

To save a Template in order to reuse it in future sessions do the following:

- To save your XML Template click on the save button and select the destination folder.
Note: Sometimes the save icon can be disabled, this is because the XML Template is empty. You must add some field before you can save it.

2.2.3.2.2.6 Adding XML Template Field

To add XML Template fields manually by selecting an area, right-clicking and selecting Add XmlField. Then click on the added field and customize it using the Object Inspector.
2.2.3.2.2.7 Adding XML Template Table

You can also create XML Template Tables.

- Select the area on which you want to create the table and right click on it.
The generated table will look like this.
Note: The number of columns are detected automatically. If the result is not what you expected, you can customize them to your wish.

- Left click on the table to customize it with the Object Inspector.
**Deleting the Table**

To delete the Table you have to do the inverse; by performing this action the XML Template will automatically be updated.

---

2.2.3.2.2.8 Adding custom XML Template Areas

The addition of a custom area to your template gives you the possibility to produce by yourself its inner XML at runtime.

To add a custom XML Template Area do the following:

- Select the area.
3 Host Integration Pack Programming

3.1 Host Integration Pack's component layers

TN BRIDGE Integration Pack design and run-time are implemented with .NET native components separated in three layers, according its purpose:

- **Mainframe Access Layer**
- **Emulation Layer**
- **Helper Controls**
3.2 Accessing to the Mainframe

Accessing to the mainframe with TN BRIDGE Host Integration Pack involves the use of the main communication components: the Tn3270 and Tn5250 components. They expose a set of properties, methods and events to handle the exchange of data.

- **Tn3270 Component**

  Implements the client TN3270E communication protocol. It connects to the telnet 3270 server that is included in the TCP/IP package for IBM S/3XX Mainframes (if it has TCP/IP support) or to an external gateway TN3270-SNA like Microsoft SNA Server. It supports connecting to a LU pool or specific LU Name. Supports extended Data Stream and IBM 3278 mod 2,3 and 4 emulation.

- **Tn5250 Component**

  Implements the client TN5250 communication protocol. It connects to the telnet 5250 server that is included in the TCP/IP package for the IBM AS/400 (if it have TCP/IP support) or to an external gateway TN5250-SNA like Microsoft SNA Server. Supports 24x80 and 27x132 terminal types.

Next, we'll learn about the main programming tasks needed to get connected and
exchange data with the mainframe.

3.2.1 Main Programming Tasks

Host Integration Pack provides methods, properties and events for connecting to the mainframe, accessing to screen data and fields, sending data and closing the connection. Also, provides means for making these calls in synchronous or asynchronous fashion.

- **Connecting/Disconnecting** explains how we can programatically get connected/disconnected to the mainframe both in synchronous and asynchronous fashion.
- **Processing Screen Data** shows the methods we can use to get access to screen data.
- **Sending Data** shows the methods we can use to fill input fields and send them along with a function key.
- **Handling Events** gives an explanation of the events that Tn components fire reacting to the different real events carried by communication with the mainframe.
- **Waiting Methods** shows the different alternatives for handling events in synchronous fashion.
- **Using Tn Pools** explain how we can acquire Tn objects from a named Pool.
- **Programming with XML** explain how we build an XML bridge between our application and the mainframe.

3.2.1.1 Connecting/Disconnecting

Host Integration Pack provides two methods for connecting up to the mainframe and one for disconnecting.

- **Synchronous Connection** shows how we can get connected in synchronous fashion.
- **Asynchronous Connection** shows how we can get connected in asynchronous fashion.
- **Disconnection** shows how to terminate the connection.

3.2.1.1.1 Synchronous Connection

This method try to establish the connection to the specified Host and Port. This call blocks the code execution, while waits until the connection is succesfully established or the specified timeout expires.

The Host and Port properties must be set for the connection to be successfully established.
\[ \text{[Boolean]} = \text{tnxxxx.Connect}(\text{Timeout As Integer}) \]

**Delphi**

\[ \text{[Boolean]} := \text{tnxxxx.Connect}(\text{Timeout:Integer}); \]

The following graphic shows the sequence of actions corresponding to the synchronous Connect method:

---

### Examples:

**VB:**

```vbnet
Private Sub cmdConnect_Click()
    telnet.Host = "clemson.clemson.edu"
    If telnet.Connect(10000) Then
        MessageBox.Show("Connected!")
    End If
End Sub
```

**Delphi:**

```delphi
procedure TForm1.Button1Click(Sender: TObject);
begin
    tnb3270.Host := 'clemson.clemson.edu';
    if tnb3270.Connect(10000) then
        ShowMessage('Connected!');
end;
```

---

### 3.2.1.1.2 Asynchronous Connection

This method tries to establish the connection to the specified Host and Port. This call **does not** block the code execution, so you should use the OnConnect and
OnConnectionFail events to determine whether the connection is successfully established or not. Alternatively, you can make a call to WaitForConnect synchronous method.

The Host and Port properties must be set for the connection to be successfully established.

**VB**

```vbnet
tnxxx.Connect()
```

**Delphi**

```delphi
tnxxx.Connect;
```

The following graphic shows the sequence of actions corresponding to the asynchronous Connect method:

### Examples:

**VB:**

```vbnet
Private Sub cmdConnect_Click()
    telnet.Host = Edit1.Text
    telnet.Port = 23
    telnet.Connect()
End Sub
```

```vbnet
Private Sub telnet_OnConnect()
    MsgBox("Connected")
End Sub
```

```vbnet
Private Sub telnet_OnConnectionFail()
    MsgBox("Connection failed")
End Sub
```
Delphi:

```delphi
procedure TForm1.Button1Click(Sender: TObject);
begin
  tnb3270.Host := Edit1.Text;
  tnb3270.Port := 23;
  tnb3270.Connect;
end;

procedure TForm1.tnb3270OnConnect(Sender: TObject);
begin
  ShowMessage('Connected');
end;

procedure TForm1.tnb3270OnConnectionFail(Sender: TObject);
begin
  ShowMessage('Connection failed');
end;
```

3.2.1.1.3 Disconnection

This method terminates the current connection.

```
VB
  tnxxxx.Disconnect()

Delphi
  tnxxxx.Disconnect;
```

The following graphic shows the sequence of actions corresponding to the Disconnect method:

![Diagram showing the sequence of actions for Disconnection](image)

Examples:

```
VB:
  Private Sub cmdDisconnect_Click()
```
3.2.1.2 Processing Screen data

The access to the screen data can be done through GetText method call or using the HostFields collection.

- **Using GetText** shows how we can read screen data using the GetText method.
- **Using HostFields** explain about HostFields collection and its use.

### 3.2.1.2.1 Using GetText

The GetText method is the simplest way to read data from the screen buffer.

**VB**

```vbnet
String = tnxxxx.GetText (StartPos As Integer, [EndPos As Integer], [Attr As Boolean = False], [Eab As Boolean = False])
String = tnxxxx.GetText (Row As Integer, Col As Integer, Len as Integer, [Attr As Boolean = False], [Eab As Boolean = False])
```

**Delphi**

```delphi
String := tnxxxx.GetText (StartPos : Integer, [EndPos : Integer], [Attr : Boolean = False], [Eab : Boolean = False])
String := tnxxxx.GetText (Row : Integer, Col : Integer, Len : Integer, [Attr : Boolean = False], [Eab : Boolean = False])
```

Start position and the end position of the text buffer to retrieve can be set using StartPos and EndPos parameters or Row,Col and Len parameters.

**Examples:**

**VB:**

```vbnet
Private Sub telnet_OnScreenChange()
' this code gets the Text from position 150
Dim s As String = telnet.GetText(150)
...
' this code gets the Text from position 150 to 200
Dim s As String = telnet.GetText(150,200)
...
' this code gets the Text from row 15 column 4 with a lenght of 20
```
Dim s As String = telnet.GetText(15,4,20)
End Sub

Delphi:

procedure TForm1.Tnb3270E1ScreenChange(Sender: TObject);
var s: string;
begin
  // this code gets the Text from position 150
  s := Tnb3270E1.GetText(150);
  ...
  // this code gets the Text from position 150 to 200
  s := Tnb3270E1.GetText(150,200);
  ...
  // this code gets the Text from row 15 column 4 with a length of 20
  s := Tnb3270E1.GetText(15,4,20);
end;

3.2.1.2.2 Using HostFields

The mainframe screen arrives in a buffer with a block of data: the data-stream. This data-stream is made up of a succession of orders, attributes and data.

The screen buffer is made up of fields. Each field is delimited by an attribute and the next one.

TN BRIDGE read these fields and collect them into the HostFields collection. Fields in HostFields collection can be accessed by index or name. The name is built up with its screen coordinates as follows: \textit{R[row number]C[column number]} For example, the field at row 4 column 30 in the screen is identified as \textit{R4C30}.

Examples:

VB:

Private Sub telnet_OnScreenChange()
  ' this code gets the Data in the Field located at Row 10 Column 2
  Dim s As String = telnet.HostFields("R10C2").Data
  ...
  ' this code gets the Data in the Field 10
  Dim s As String = telnet.HostFields[10].Data
End Sub

Delphi:
procedure TForm1.Tnb3270E1ScreenChange(Sender: TObject);
var s:string;
begin
  // this code gets the Data in the Field located at Row 10 Column 2
  s := Tnb3270E1.HostFields["R10C2"].Data;
  ...
  // this code gets the Data in the Field 10
  s := Tnb3270E1.HostFields[10].Data;
  ...
end;

3.2.1.3 Sending Data

Sending data to the mainframe can be split into two steps: filling the edit data buffer and sending all the modified data to the mainframe along with a function key.

- **Using Type method** explains how to fill input fields by simulating the keyboard entry.
- **Using EditFields collection** explains the use of EditFields collection.
- **The Press and PressAndWait methods** explain how to use unblocking or blocking methods to send data and function keys to the mainframe.

3.2.1.3.1 Using Type method

The Type method can be used to send key sequences to the mainframe starting from the current cursor position.

**VB**

`tnxxxx.Type (Keys As String)`

**Delphi**

`tnxxxx.Type (Keys : String);`

By using special codes you can send several special keys. These codes consist of an escape character ("@") and a mnemonic code that corresponds to the supported function.

Type method can also make entered data to be sent along with an AID key (Attention Identifier key), avoiding the use of Press method.

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td></td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td></td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td></td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td></td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td></td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td></td>
<td>Erase Field</td>
</tr>
</tbody>
</table>
### Examples:

**VB:**

```vbnet
Private Sub SendUserAndPassword()
    ' @T is translated to TAB key
    telnet.Type(userId + "@T" + password)
    telnet.PressAndWait(Enter)
End Sub
```

**Delphi:**

```delphi
procedure SendUserAndPassword(userId, password: String);
begin
    // @T is translated to TAB key
    tnb3270.Type(userId + '@T' + password);
    tnb3270.PressAndWait(Enter);
end;
```

### 3.2.1.3.2 Using EditFields Collection

The EditFields collection contain a reference to all the fields in HostFields collection with the unprotected attribute set to true. These fields can be accessed in order to modify its data, which later will be sent to the mainframe along with a function key (AID key).

**VB**

```vbnet
[Field = tnxxxx.EditFields (Index As Integer)]
[Field = tnxxxx.EditFields (Index As String)]
```
Delphi

\[\text{Field := tnxxxx.EditFields (Index : Integer);}\]
\[\text{Field := tnxxxx.EditFields (Index : String);}\]

Examples:

**VB**

Private Sub cmdLogin_Click()
    ' this code sets userId in Field located at Row 10 Column 2
telnet.EditFields["R10C2"].Data = "puser"
    ' this code sets the password in Field located at Row 12 Column 2
telnet(EditFields["R12C2"].Data = "k2156"
    ' this code sends the ENTER Attention Identifier Key along with all modified EditFields
telenet.Press("Enter")
End Sub

**Delphi**

procedure TForm1.LoginClick(Sender:TObject);
begin
    ' this code sets userId in Field located at Row 10 Column 2
tnb3270.EditFields['R10C2'].Data := 'puser';
    ' this code sets the password in Field located at Row 12 Column 2
tnb3270(EditFields['R12C2'].Data := 'k2156';
    ' this code sends the ENTER Attention Identifier Key along with all modified EditFields
    tnb3270.Press('Enter');
end;

3.2.1.3.3 The Press and PressAndWait methods

In a real terminal, the typed data is sent to the mainframe upon pressing one of the keys known as Attention Identifier keys (AID). These keys act as function keys that are sent along with the typed data. In Host Integration Pack, you can use the Press and PressAndWait methods to simulate this action.

- **The Press method** explains how you can send an AID key in unblocking fashion
- **The Press and PressAndWait methods** explains how you can send an AID Key in a blocking fashion.

3.2.1.3.3.1 The Press method

The Press method sends an Attention Identifier Key along with the modified fields. Modified fields can be input fields (unprotected) or sometimes protected fields having the property Modified set to True.

**VB**

\[\text{tnxxxx.Press (aidKey as AidKey)}\]
\[\text{tnxxxx.Press (aidKey As String)}\]
Delphi

```
tnxxxx.Press (aidKey : AidKey);
tnxxxx.Press (aidKey : String);
```

The following graphic shows the sequence of actions corresponding to the asynchronous Press method:

```
Note: The execution of the Type method is stopped when an AID Key is sent to the Host.

Examples:

VB:
```
Private Sub cmdLogin_Click()
    ' this code sets userId in Field located at Row 10 Column 2
tenet.EditFields("R10C2").Data = "puser"
    ' this code sets the password in Field located at Row 12 Column 2
tenet.EditFields("R12C2").Data = "k2156"
    ' this code sends the ENTER Attention Identifier Key along with all modified EditFields
tenet.Press("Enter")
End Sub
```

Delphi:
```
procedure TForm1.LoginClick(Sender:TObject);
begin
    // this code sets userId in Field located at Row 10 Column 2
    tnb3270.EditFields['R10C2'].Data := 'puser';
    // this code sets the password in Field located at Row 12 Column 2
    tnb3270.EditFields['R12C2'].Data := 'k2156';
    // this code sends the ENTER Attention Identifier Key along with all modified EditFields
    tnb3270.Press('Enter');
end;
```
### 3.2.1.3.3.2 The PressAndWait method

The PressAndWait method acts like Press method, but it blocks the code execution waiting until the system gets unlocked.

**VB**

```vb
tnxxxx.PressAndWait (aidKey as AidKey)
```

```vb
tnxxxx.PressAndWait (aidKey As String)
```

The following graphic shows the sequence of actions corresponding to the synchronous PressAndWait method:

![Diagram](image)

**Examples:**

**VB:**

```vb
Private Sub cmdLogin_Click()
    ' this code sets userId in Field located at Row 10 Column 2
tenet.EditFields("R10C2").Data = "puser"
    ' this code sets the password in Field located at Row 12 Column 2
tenet.EditFields("R12C2").Data = "k2156"
    ' this code sends the ENTER Attention Identifier Key along with all modified EditFields
tenet.Press("Enter")
End Sub
```

**Delphi:**

```delphi
procedure TForm1.LoginClick(Sender: TObject);
begin
    // this code sets userId in Field located at Row 10 Column 2
tnb3270.EditFields['R10C2'].Data := 'puser';
    // this code sets the password in Field located at Row 12 Column 2
    tnb3270.EditFields['R12C2'].Data := 'k2156';
    // this code sends the ENTER Attention Identifier Key along with all modified EditFields
tnb3270.Press('Enter');
end
```
3.2.1.4 Handling Events

Tn events are specially important in asynchronous programming. A typical case is a custom terminal emulator, where the screens and state info must be reflected in the user interface.

The Tn events are:

- **OnConnect**: Fires after a successfully connection to the mainframe is established.
- **OnConnectFail**: Fires after the connection to the mainframe fails.
- **OnDisconnect**: Fires after a disconnection with the mainframe.
- **OnSessionState**: Fires when a session-state change takes place.
- **OnSystemLock**: Fires when a terminal changes to a system locked state.
- **OnSystemUnLock**: Fires when a terminal changes to a system unlocked state.
- **OnScreenChange**: Fires after a new data screen has arrived.
- **OnSendAid**: Fires when an AID key is about to be sent (before is sent).

Some code examples:

- You can monitor the connection status with the following events:

  **VB**:
  ```vb
  Private Sub telnet_OnConnect()
    MessageBox.Show("Connected to the Host!");
  End Sub

  Private Sub telnet_OnConnectionFail()
    MessageBox.Show("Connection Failed!");
  End Sub

  Private Sub telnet_OnDisconnect()
    MessageBox.Show("Disconnected from the Host!");
  End Sub
  ```

  **Delphi**:
  ```delphi
  procedure TForm1.Tnb3270E1Connect(Sender: TObject);
  begin
    ShowMessage('Connected to the Host!');
  end;

  procedure TForm1.Tnb3270E1ConnectionFail(Sender: TObject);
  begin
    ShowMessage('Connection Failed!');
  end;

  procedure TForm1.Tnb3270E1Disconnect(Sender: TObject);
  begin
    ShowMessage('Disconnected from the Host!');
  end;
  ```
In these events you can display or use the host data

**VB:**

```vbnet
Private Sub telnet_OnScreenChange()
    MessageBox.Show("A new screen has been received");
End Sub

Private Sub telnet_OnSystemUnLock()
    MessageBox.Show("The latest screen has arrived");
End Sub
```

**Delphi:**

```delphi
procedure TForm1.Tnb3270E1.ScreenChange(Sender: TObject);
begin
    ShowMessage('A new screen has been received');
end;

procedure TForm1.Tnb3270E1.SystemUnLock(Sender: TObject);
begin
    ShowMessage('The latest screen has arrived');
end;
```

These events controls when you are able to send data to the mainframe

**VB:**

```vbnet
Private Sub telnet_OnSystemUnLock()
    MessageBox.Show("We can send data to the host now");
End Sub

Private Sub telnet_OnSystemLock()
    MessageBox.Show("We can't send data at this moment!");
End Sub
```

**Delphi:**

```delphi
procedure TForm1.Tnb3270E1.SystemUnLock(Sender: TObject);
begin
    ShowMessage('We can send data to the host now');
end;

procedure TForm1.Tnb3270E1.SystemLock(Sender: TObject);
begin
    ShowMessage('We can't send data at this moment!');
end;
```

Now you know how to connect to the host and handle the communication events. It's time to process the host data:

**VB:**

```vbnet
Private Sub telnet_OnSystemUnLock()
    'At this time we can send data to the host
    'First we check if we are in the log-on screen

    If (telnet.GetText(15,1,telnet.Cols).Substring(10,8) = "UserId:" ) Then
```
'Now we fill in the input fields
__ telnet.EditFields[0].Data = "MyUserId"
__ telnet.EditFields[1].Data = "MyPassword"
'and send them to the host with the Enter Key!
__ telnet.Press(Enter)

'Else If

End If

End Sub

Delphi:

procedure TForm1.Tnb3270E1SystemUnlock(Sender: TObject);
var
Field: TTNBField;

begin
  // At this time we can send data to the host
  // First we check if we are in the log-on screen

  if SearchBuf(PChar(Tnb3270E1.ScreenRow[15]),
              length(Tnb3270E1.ScreenRow[15]+1), 10, 8, 'UserId:')<> nil then
    begin

    // Now we fill in the input fields
    Tnb3270E1.EditFields[0].Data := 'MyUserId';
    Tnb3270E1.EditFields[1].Data := 'MyPassword';

    // and send them to the host with the Enter Key!
    Tnb3270E1.Press(Enter);
    end
  else begin

  end;
end;

3.2.1.5 Waiting Methods

There are several waiting methods that allow to wait for specifics events or event sequence, blocking the code execution until the event is raised or the operation times out. Main methods are:

VB

[Boolean] = tnxxxx.WaitForConnect ([Timeout As Integer])
[Boolean] = tnxxxx.WaitForScreen ([Timeout As Integer])
[Boolean] = tnxxxx.WaitForUnlock ([Timeout As Integer])
[Boolean] = tnxxxx.WaitFor (StringValue As String, [Timeout As Integer])
[Integer] = tnxxxx.WaitFor (StrValues As Array of String, [Timeout As Integer])

Delphi

[Boolean] := tnxxxx.WaitForConnect ([Timeout : Integer]);
[Boolean] := tnxxxx.WaitForScreen ([Timeout : Integer]);
[Boolean] := tnxxxx.WaitForUnlock ([Timeout : Integer]);
[Boolean] := tnxxxx.WaitFor (StringValue : String, [Timeout : Integer]);
[Integer] := tnxxxx.WaitFor (StrValues : Array of String, [Timeout : Integer]);

WaitForConnect waits until the telnet connection is successfully established.
WaitForScreen waits until a new screen arrives after the connection is established or an AID key was sent.
**WaitForUnlock** waits until the host system turns to an unlocked state.
**WaitFor** waits for a screen containing the specified string or strings.

The following graphic shows the sequence of actions corresponding to the WaitFor method:

![Diagram](image)

**Examples:**

**VB:**
```vba
Private Sub cmdGetInfo_Click()
    telnet.Host = Edit1.Text
    If (telnet.Connect(10000) Then
        telnet.Type("Anonymous")
        telnet.PressAndWait("ENTER")
        telnet.Type("srch "+Edit2.Text+"@E")
        telnet.WaitFor("Library")
        Edit3.Text = telnet.GetText(20,5,30)
        telnet.Disconnect()
    End If
End Sub
```

**Delphi:**
```delphi
procedure TForm1.GetInfoClick(Sender: TObject);
begin
    TNB32701.Host := Edit1.Text;
    if TNB32701.Connect(10000) then
    begin
        TNB32701.Type('Anonymous');
        TNB32701.PressAndWait('ENTER');
        TNB32701.Type('srch '+Edit2.Text+'@E');
        TNB32701.WaitFor('Library');
    end;
end;
```
3.2.1.6 Using Tn Pools

TN BRIDGE Host Integration Pack allows persisting Tn3270/5250 objects in a Pools that are available in the whole application domain.

The method used to acquire a Tn object from the pool is:

**VB**

```vbnet
tnbxxxx = tnbxpool.Acquire([sessionId as String])
```

**Delphi**

```pascal
tnbxxx = Tnbxxxx.FromPool(poolId : String,[sessionId : String],[InactivityTimeout : Integer]);
```

This method is static (shared) and it returns a Tn from the Pool with the name poolId and optionally identified by SessionId parameter. In case of no Tn object is available, it returns a new Tn object which is also added to the pool.

The Tn object must be released by using the method:

**VB**

```vbnet
tnbxpool.Release(tnobject)
```

**Delphi**

```pascal
tnbxxx.Release;
```

Persisting telnet objects in pools allows to:

- **Reusing the connection** in a disconnected environment like ASP.NET
- **Keeping Tn objects ready** for their reuse.

3.2.1.6.1 Reusing the connection

In a disconnected environment, the Tn Pool allow to persist the Tn object on a session basis.
Upon creation, Tn object generates an unique identifier that can be accessed through the SessionId property.

**Examples:**

**VB:**

```vbnet
Private Function GetNewTn3270 As String
    Dim Pool As TTNBXPOOL
    Dim telnet As TTNB3270X

    Set Pool = new TTNBXPOOL
    Pool.Id = "Clemson"
    Pool.Timeout = 3
    Set telnet = Pool.Acquire
```
GetNewTn3270 = telnet.SessionId;
Pool.Release;
end;

Private Sub Transact(id As string);
  Dim Pool As TTNBXPOOL
  Dim telnet As TTNB3270X

  Set Pool = new TTNBXPOOL
  Pool.Id = "Clemson"
  Pool.Timeout = 3
  Set telnet = Pool.Acquire(id)
  if not telnet.IsConnected() then
    telnet.Host = 'clemson.clemson.edu'
    telnet.Port = 23
    telnet.Connect(10000)
  end if

  ...
  Pool.Release telnet
End Sub

Delphi:
function TForm1.GetNewTn3270:string;
var telnet:TTnb3270;
begin
  telnet := TTnb3270E.FromPool('Clemson');
  try
    result:= telnet.SessionId;
  finally
    telnet.Release;
  end;
end;

procedure TForm1.Transact(id:string);
var telnet:TTnb3270;
begin
  telnet := TTnb3270E.FromPool('Clemson',id);
  try
    if not telnet.IsConnected() then
      begin
        telnet.Host := 'clemson.clemson.edu';
        telnet.Port := 23;
        telnet.Connect(10000);
        ...
      end;
  finally
    telnet.Release;
  end;
end;

3.2.1.6.2 Keeping Tn objects ready

Many times, having these objects connected and logged in into a mainframe application is the common start and ending point for making a mainframe transaction (CICS transactions are a clear example). So maintaining Tn objects ready for their reuse is a good technique
for avoiding unnecessary delays.

In these cases you can acquire a Tn object just by passing the pool name without any object identifier. This way you will get a free Tn object to reuse.

**Examples:**

**VB:**

```vbnet
Private Sub Load()
    Dim Pool As TTNBXPOOL
    Dim telnet As TTNB3270X

    Set Pool = new TTNBXPOOL
    Pool.Id = "Clemson"
    Pool.Timeout = 3
    Set telnet = Pool.Acquire

    If Not telnet.IsConnected() Then
        telnet.Host = "clemson.clemson.edu"
        telnet.Port = 23
        telnet.Connect(1000)
    End If
    ...
    Pool.Release()
End Sub
```

**Delphi:**

```delphi
procedure TForm1.Button1Click(Sender: TObject);
var telnet:TTnb3270;
begin
    telnet := TTnb3270E.FromPool('Clemson');
    try
        if not telnet.IsConnected() then
            begin
                telnet.Host := 'clemson.clemson.edu';
                telnet.Port := 23;
                telnet.Connect(10000);
            end;
        ...
    finally
        telnet.Release;
    end;
end;
```

### 3.2.1.7 Programming with XML

Host Integration Pack allows you to build an XML bridge between your application and the mainframe. The XmlBroker component is the one that allows to produce XML code from mainframe screens and convert XML input data and send it to the mainframe.

- **Providing Generic XML** explains how we can provide or consume XML based on HostFields.
- **Providing a more friendly XML** explains how we can provide and consume custom XML, more appropriate for interoperability with other platforms and environments.
3.2.1.7.1 Providing Generic XML

The XmlBroker is the key component to get a Xml representation of the screen. Its usage can be seen in the following code snippet:

Examples:

VB:

Private Sub GetXml() As String
    Dim broker1 As TTNBXXmlBroker = new TnbXXmlBroker(telnet)
    GetXml = broker1.Xml
End Sub

Private Sub SetXml(ByVal Xml As String)
    Dim broker1 As TTNBXXmlBroker = new TnbXXmlBroker(telnet)
    broker1.Xml = Xml
End Sub

Delphi:

function GetXml:String;
var broker:TTnbXmlBroker;
begin
    broker := TTnbXmlBroker.Create(telnet);
    result := broker.Xml;
end;

procedure SetXml(Xml:String);
var broker:TTnbXmlBroker;
begin
    broker := TTnbXmlBroker.Create(telnet);
    broker.Xml = Xml;
end;

The XML code generated is as follows:

<TNBbridge>
<status>
    <direction... />
    <transaction... />
    <cursor_pos... /> 
    <session... /> 
    <state... />
    <xmlscreen... />
    <cursor_field... />
    <timestamp... />
<status />
<screen>
    <rows... />
    <cols... />
    <type... />
    <model... />
    <fields>
        <field ...
            <name... />
TNBRIDGE is the root tag and includes the status and screen tags as children.

**status:** indicates the connection status
Contains the following tags:

* direction: indicates INPUT or OUTPUT direction. When the XML is generated by the XmlBroker control it is set with OUTPUT value.
* transaction: indicates transaction identifier.
* cursor_pos: indicates cursor location.
* session: LU-LU (in session with VTAM application), noSession (not in session) or SSCPLU (in session with VTAM).
* state: indicates LOCKED or UNLOCKED state.
* xmlscreen: indicates the screen name. You must set this property using the ScreenName property.
* cursor_field: indicates the field's name where the cursor is located.
* timestamp: indicates the date the file was created.

**Screen:** indicates screen's characteristics and contains a collection of screen fields.
Contains the following tags:

* rows: indicates the number of rows of the screen.
* cols: indicates the number of columns of the screen.
* type: indicates terminal type (3270 or 5250).
* model: indicates terminal model.
* fields: contains a collection of fields.

Here you can see an example of XML Code:

```xml
<TNBRIDGE>
  <status>
    <direction>OUTPUT</direction>
    <transaction>00A8DB7000002</transaction>
    <cursor_pos>830</cursor_pos>
    <session>LU-LU</session>
    <state>UNLOCKED</state>
    <xmlscreen name="Session"/>
    <cursor_field>R11C30</cursor_field>
    <timestamp>2453360.14494451</timestamp>
  </status>
  <screen>
    <rows>24</rows>
    <cols>80</cols>
  </screen>
</TNBRIDGE>
```
<type>TN3270</type>
<model>2E</model>

<fields>
  <field name="R01C02">
    <name len="4">R1C2</name>
    <value maxlen="8" len="8">KLGLGON1</value>
  </field>

  <field name="R01C11">
    <name len="5">R1C11</name>
    <value maxlen="13" len="13">-------------</value>
  </field>

.</fields>
</screen>
</TNBRIDGE>

3.2.1.7.2 Providing a more friendly XML

Generating a more friendly XML involves the use of XML Templates. Next we'll see:

- **What is an XML Template?**
- **Using XML Templates**

3.2.1.7.2.1 What is an XML Template

An XML Template is a template file that provides information about how to build an XML file having a particular mainframe screen as data source. XML Templates are built with Development Lab. Let's see an example:

When transforming this screen to XML we would like to use tags according what we know
are labels and values according what we know that actually are values. As an example, we know that "Status of job" is a label and "ACTIVE" is its value. So, what we would like to see in translated to XML would be something like:

<StatusOfJob>ACTIVE</StatusOfJob>

With the help of Development Lab and our own help, we can define a template to instruct TN BRIDGE to translate the mainframe screen to a readable XML.

Here we can see how Labels and values were separated by Development Lab and this is the XML we get as result:

```xml
<Screen>
  <Title>Display Job Status Attributes</Title>
  <System>TS400</System>
  <Job>QPADEV002V</Job>
  <User>GRICARDI</User>
  <Number>297131</Number>
  <StatusOfJob>ACTIVE</StatusOfJob>
  <CurrentUserProfile>GRICARDI</CurrentUserProfile>
  <JobUserIdentity>GRICARDI</JobUserIdentity>
  <SetBy>*DEFAULT</SetBy>
  <EnteredSystem>
    <Date>05/06/05</Date>
    <Time>08:21:47</Time>
  </EnteredSystem>
  <Started>
    <Date>05/06/05</Date>
    <Time>08:21:47</Time>
  </Started>
  <Subsystem>BASE</Subsystem>
    <SubsystemPoolID>2</SubsystemPoolID>
  </Subsystem>
  <TypeOfJob>INTER</TypeOfJob>
  <SpecialEnvironment>*NONE</SpecialEnvironment>
  <ProgramReturnCode>1</ProgramReturnCode>
</Screen>
```
3.2.1.7.2.2 Using XML Templates

How do we use XML Templates? First we need to create XML templates for our mainframe screens. These templates are saved into a common directory. Once we have these templates ready we can use the XmlBroker component and XmlTemplate class as follows:

Examples:

VB:

Private Sub ProduceXml() As String
'first we instanciate the XmlBroker
Dim broker1 As TTNBXXmlBroker = new TnbXXmlBroker(telnet)

'then we change its XML producer by assigning an XmlTemplate object
broker1.Producer = new TTNBXXmlTemplate("c:\My Templates")

'now, the returned Xml will be produced by the XmlTemplate object and
'the corresponding template file.
ProduceXml = broker1.Xml
End Sub

Delphi:

procedure ProduceXml:String;
var broker:TTnbXmlBroker;
begin
//first we instanciate the XmlBroker
broker := TTnbXmlBroker.Create(tn3270);

//then we change its XML producer by assigning an XmlTemplate object
broker.Producer = XmlTemplate.Create('c:\My Templates');

//now, the returned Xml will be produced by the XmlTemplate object and
//the corresponding template file.
result := broker.Xml;
end;

We need to tell to XmlTemplate object the directory where the XML template files are located. Then, when the XmlBroker needs to produce a new XML, it ask to the producer (in this case XmlTemplate) if it can produce XML content from the current screen. Then, XmlTemplate object looks for a template that matches with the current screen and returns true or false according to it. Next, in case it returned true, it is asked to produce the XML content which finally will be the one delivered by XmlBroker's Xml property; in case it returned false, the XmlBroker uses the default producer to generate XML content, returning a generic XML based on HostFields.
3.3 Working with the Display

3.3.1 Terminal Emulator Controls

The TN BRIDGE Emulation Layer is made up of 5 Terminal Emulator components. These components implement the main features for a terminal emulator program like: Display Screen, Status Bar, Keyboard Handling, Macro Sequences and Profiles.

- **Emulation Screen: TnbXEmulator component**
  
  Implements an emulation panel. This panel shows host data and accepts input data to be sent to the mainframe. The displayed data is formatted according to the data stream format (3270 or 5250).

- **Emulation Status Bar: TnbXStatusBar component**
  
  Implements an emulation status bar. It works joined to the TNBEmulator component, complementing it.

- **Keyboard Handling: TnbXAidHook component**
  
  Process the keyboard and maps to Emulation Function Keys and Attention Identifier Keys. The Emulation Function Keys are those which are processed locally to manage the emulator behavior for input data (For example: Tab key, Cursor movement keys, Home key, etc.). The Attention Identifier Keys are those which provoke sending data and the properly Attention Identifier Code (For example: Enter key, PF01 key, Clear key, etc.).

- **Macro Sequences: TnbXMacro component**
  
  Implements the recording and playing mainframe navigation sequences to automate tasks.

- **Profiles: TnbXProfiles component**
  
  Implements a common interface for saving and loading profiles like screen emulation styles, host connection profiles and macro sequences.

3.3.2 Main Programming Tasks

For using the Terminal Emulator components the main tasks are:

- Drop a Tn3270 or Tn5250 Component on your form.
- Drop to your form the following components: Display, StatusBar and Keyboard.
- Set the Telnet property for Emulator and StatusBar components to the telnet component (Tn3270 or Tn5250). You can set it at run-time using the following code in the Load or Create event of the form:

  **VB:**
  
  ```vb
  Private Sub Form_Load()
  TnbXDisplay1.TelnetControl = TnB3270X1
  ```
TNBXStatusBar1.TelnetControl = TNB3270X1
...
End Sub

Delphi:
procedure TForm1.Load(Sender:TObject);
begin
  tnbEmulator.TnbCom := tnb32701;
  tnbStatusBar.TnbCom := tnb32701;
  ...
end;

• Set the *EmulatorComponent property* from StatusBar component to the Display dropped in the form.

  VB:
  TNBXStatusBar1.EmulatorControl = TNBXAidHook1

  Delphi:
  tnbStatusBar.TnbEmulator := tnbDisplay1;

• Set *KeyboardComponent property* from Emulator component to the Keyboard dropped in the form.

  VB:
  TNBXEmulator1.AidHookControl = TNBXAidHook1

  Delphi:
  tnbEmulator.TnbAidHook = tnbAidHook1;

• Now, all what you have to do is fill in the *Host property* for the Telnet component chosen, with your Mainframe DNS name or TCP/IP address and execute the *Connect method*. For example:

  VB:
  Private Sub Button1_Click()
    'In the edit box you have to type the DNS name or
    'TCP/IP address of the mainframe you want to connect to.
    TNB3270X1.Host = Edit1.Text
    TNB3270X1.Connect()
    ...
  End Sub

  Delphi:
  procedure TForm1.Button1Click(Sender:TObject);
  begin
    // In the edit box you have to type the DNS name or
    // TCP/IP address of the mainframe you want to connect to.
    tnb3270.Host := Edit1.Text;
    tnb3270.Connect;
    ...
  end;
Now you have a complete terminal emulator.

4 Component Reference for ActiveX and Delphi

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack ActiveX and Delphi components.

- ActiveX Components
- Delphi Components

4.1 ActiveX Components

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack ActiveX components.

- Syntax Conventions
- Mainframe Access Controls
- Terminal Emulator Controls
- Helper Controls
- Trace Services

4.1.1 Syntax conventions

The following text formats are used throughout the Components Reference:

For each property and method:

**Italic text** Denotes an object, in this case an Integration Pack control.

**Bold text** Denotes a property or a method names.

`[= value]` Denotes values to be assigned.

`[value = ]` Denotes return values.

`(TimeOut)` Denotes a parameter to be passed to a method.

`([TimeOut])` Denotes an optional parameter to be passed to a method.

4.1.2 Mainframe Access Controls

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack ActiveX Mainframe Access Controls.

4.1.2.1 TNB5250X/TNB3270X Reference

4.1.2.1.1 TNB5250X/TNB3270X Controls

Both controls share a common programming interface, except for a few properties that are specific to one control.

**Properties**

- AIDKey
- CodePage
- Cols
- ConnectionName
- ConversionTable
- Debug
- DebugDir
- DeviceName
- EditFields
- ExcludeEmptyFields
- Extended
- Host
- HostFields
- IsExtended
- KeepAlive
- LastErrMsg
- Password
- Port
- Profiles
- Rows
- ScreenRow
- SessionState
- SSL
- SSLAcceptExpired
- SSLAcceptInvalid
- SSLAcceptInvalidCA
- SSLAcceptNotYetValid
- SSLAcceptSelfSigned
- SSLCertFile
- SSLCertPassword
- SSLDisplay
- SSLEnableDialogs
- SSLKeyFile
- SSLMethod
- SSLRootCertFile
- SubKey
- TerminalType
- UserId
- XLockDelay

Methods

- AsVariant
- ClassNameIs
- CloseEditor
- Connect
- Disconnect
- GetScreenRowEx
- GetScreenText
- IsConnected
- LoadFromXMLFile
- SaveToXMLFile
- SendAid
- SendKeys
- ShowEditor
Events

- **OnConnect**
- **OnConnectionFail**
- **OnDisconnect**
- **OnMessageWaitingOff**
- **OnMessageWaitingOn**
- **OnScreenChange**
- **OnSendAid**
- **OnSessionState**
- **OnSystemLock**
- **OnSystemUnlock**

### 4.1.2.1.2 Properties

#### 4.1.2.1.2.1 AIDKey

Sets the Attention Identifier Key and sends the modified fields to the Mainframe. Modified fields can be input fields (unprotected) or, sometimes protected fields, having the property Modified set to True.

**Visual Basic Syntax**

\[TNBnnnnX.AIDKey \[= \text{TNBXAid}\]\]

**Delphi Syntax**

\[TNBnnnnX.AIDKey \[:= \text{TNBXAid}\];\]

It can take any of TNBAidKey constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>akPF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>akPF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>akPF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>akPF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>akPF12</td>
<td>PF12 key</td>
</tr>
</tbody>
</table>
See also [TNBXAid](#) constants.

4.1.2.1.2.2 Codepage

Specifies the internal ASCII-EBCDIC conversion table.

**Visual Basic Syntax**

\[ TNBnnnnX.CodePage \[ = Integer \] \]

**Delphi Syntax**

\[ TNBnnnnX.CodePage \[ := Integer \]; \]

**Remarks**

It can take any of the Code Pages constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia   = 037</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium     = 037</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada      = 037</td>
<td>Canada</td>
</tr>
<tr>
<td>cpCzechRepublic = 870</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cpDenmark     = 277</td>
<td>Denmark</td>
</tr>
<tr>
<td>cpFinland     = 278</td>
<td>Finland</td>
</tr>
<tr>
<td>cpFrance      = 297</td>
<td>France</td>
</tr>
<tr>
<td>cpGermany     = 273</td>
<td>Germany</td>
</tr>
<tr>
<td>cpGreece</td>
<td>Greece</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------</td>
</tr>
<tr>
<td>cpGreece2</td>
<td>Greece</td>
</tr>
<tr>
<td>cpItaly</td>
<td>Italy</td>
</tr>
<tr>
<td>cpLatinAmerica</td>
<td>Latin America</td>
</tr>
<tr>
<td>cpNetherlands</td>
<td>Netherlands</td>
</tr>
<tr>
<td>cpNorway</td>
<td>Norway</td>
</tr>
<tr>
<td>cpPoland</td>
<td>Poland</td>
</tr>
<tr>
<td>cpPortugal</td>
<td>Portugal</td>
</tr>
<tr>
<td>cpSpain</td>
<td>Spain</td>
</tr>
<tr>
<td>spSweden</td>
<td>Sweden</td>
</tr>
<tr>
<td>cpSwitzerlandGerman</td>
<td>Switzerland (German)</td>
</tr>
<tr>
<td>cpSwitzerlandFrench</td>
<td>Switzerland (French)</td>
</tr>
<tr>
<td>cpUnitedKingdom</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>cpUnitedStates</td>
<td>United States</td>
</tr>
<tr>
<td>cpInternational</td>
<td>International</td>
</tr>
</tbody>
</table>

Default value is cpUnitedStates.

See also [ConversionTable](#) property and Code Pages constants.

### 4.1.2.1.2.3 Cols

Return the total number of columns of the current terminal type.

**Visual Basic Syntax**

```vbnet
[Integer =] TNBnnnnX.Cols
```

**Delphi Syntax**

```delphi
[TNBnnnnX.ConnectionName := String];
```

**Remarks**

The Cols property may return 80 or 132 values depending on the terminal type chosen.

See also [Rows](#) and [TerminalType](#) properties.

### 4.1.2.1.2.4 ConnectionName

Sets/gets the connection name under which will be stored the connection.

**Visual Basic Syntax**

```vbnet
TNBnnnnX.ConnectionName [= String]
```

**Delphi Syntax**

```delphi
TNBnnnnX.ConnectionName [= String];
```

The default values are "TNB3270E" for TNB3270X Control and "TNB5250" for TNB5250X
Remarks

This property is valid only if a ProfilesControl property has been assigned to a TNBXProfiles Control.

See also ProfilesControl property and TNBXProfiles Component.

4.1.2.1.2.5 ConversionTable

Specifies an external ASCII-EBCDIC conversion table to be loaded from a file. If no one is specified, an internal conversion table is used, according to CodePage property setting.

Visual Basic Syntax

TNBnnnnX.ConversionTable [= String]

Delphi Syntax

TNBnnnnX.ConversionTable [= String];

Remarks

The file specified must exist and it might contain a valid conversion table format.

The format of the conversion table is:

[Ascii-To-Ebcdic]
Ascii hexadecimal code 1 = Ebcdic hexadecimal code 1
Ascii hexadecimal code 2 = Ebcdic hexadecimal code 2
... Ascii hexadecimal code n = Ebcdic hexadecimal code n

[Ebcdic-To-Ascii]
Ebcdic hexadecimal code 1 = Ascii hexadecimal code 1
Ebcdic hexadecimal code 2 = Ascii hexadecimal code 2
... Ebcdic hexadecimal code n = Ascii hexadecimal code n

See also CodePage property.

4.1.2.1.2.6 Debug

Allows the generation of trace information for debugging purposes. The debugging can help you to track down both, errors and logic information.

Visual Basic Syntax

TNBnnnnX.Debug [= Boolean]
Delphi Syntax

TNBnnnnX. Debug[:, Boolean];

The following are possible values for Debug property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Enables debug information.</td>
</tr>
<tr>
<td>False</td>
<td>Disables debug information.</td>
</tr>
</tbody>
</table>

Remarks

By default, the debug file is saved in the application directory. You can change the directory by setting the DebugDir property.

Default value is False.

See also DebugDir property.

4.1.2.1.2.7 DebugDir

Specifies the directory for debug files.

Visual Basic Syntax

TNBnnnnX. DebugDir [= String]

Delphi Syntax

TNBnnnnX. DebugDir[:, String];

Remarks

By default, the debug file is saved in the application's directory. You can change it by setting this property to another directory. This directory must exists, otherwise, no debug information will be generated and an error will not occur.

The default value is the application directory.

See also Debug property.

4.1.2.1.2.8 DeviceName

Sets an specific telnet device name defined in the telnet server.

Visual Basic Syntax

TNBnnnnX. DeviceName [= String]

Delphi Syntax
Remarks

If no name is given, the telnet server will assign an available device from a public pool in case of it exists.

See also **UserId** and **Password** properties.

4.1.2.1.2.9 EditFields

Array containing the editable screen fields.

**Visual Basic Syntax**

[TNBXField] = TNBnnnnX.EditFields (Index As Variant)

**Delphi Syntax**

[TNBXField] := TNBnnnnX.EditFields (Index:Variant);

Remarks

This list includes only the unprotected **TNBXFields** of the mainframe's screen. If you specified a wrong index value, a null value is returned.

See also **HostFields** property, **TNBXFields** and **TNBXField** Classes.

4.1.2.1.2.10 ExcludeEmptyFields

Specifies to the telnet control to exclude or not host empty fields.

**Visual Basic Syntax**

TNBnnnnX.ExcludeEmptyFields [= Boolean]

**Delphi Syntax**

TNBnnnnX.ExcludeEmptyFields [:= Boolean];

Remarks

Default value is False.

See also **HostFields** property.

4.1.2.1.2.11 Extended

Specifies if the telnet 5250 extended protocol is allowed by this client.

**Visual Basic Syntax**
**TNBnnnnX.**\texttt{Extended} \[ = \text{Boolean} \]

### Delphi Syntax

\[ TNBnnnnX. Extended \[ : = \text{Boolean}; \]

### Remarks

If the host or telnet server supports the TN5250E protocol, it will try to negotiate with this client to work in extended mode. If this property is set to False, this client will deny the request from the server and both will continue in TN5250 mode. Default value is True.

See also \texttt{Host}, \texttt{Port} and \texttt{TerminalType} property.

#### 4.1.2.1.2.12 Host

TCP/IP address or DNS name of the Host to connect to. It can be a name like 'myhost.mydomain.com' or directly an IP address such as '206.155.164.20'.

### Visual Basic Syntax

\[ TNBnnnnX. Host \[ = \text{String} \]

### Delphi Syntax

\[ TNBnnnnX. Host \[ : = \text{String}; \]

### Remarks

The host name stored in the Host property must be alpha numeric, and can't contain spaces nor special characters (i.e. : " / ? \)

See also \texttt{Port} property, \texttt{Connect} method and \texttt{OnConnect} event.

#### 4.1.2.1.2.13 HostFields

Array containing the screen fields.

### Visual Basic Syntax

\[ \text{TNBXField} = TNBnnnnX. HostFields (Index As Variant) \]

### Delphi Syntax

\[ \text{TNBXField} := TNBnnnnX. HostFields [Index:Variant]; \]

### Remarks
This list includes all **TNBXFields** (protected and unprotected) of the mainframe’s screen. If you specified a wrong index value, a null value is returned.

See also **EditFields** property, **TNBXFields** and **TNBXField** classes.

### 4.1.2.1.2.14 IsExtended

Returns a boolean value indicating if the current connection uses TN5250E capabilities.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBnnnnX.IsExtended
```

**Delphi Syntax**

```delphi
[Boolean :=] TNBnnnnX.IsExtended;
```

See also **Extended** property.

### 4.1.2.1.2.15 KeepAlive

Allows to enable a keep-alive mechanism used by some telnet servers.

**Visual Basic Syntax**

```vbnet
TNBnnnnX.KeepAlive [= Boolean]
```

**Delphi Syntax**

```delphi
TNBnnnnX.KeepAlive [= Boolean];
```

**Remarks**

Default value for telnet 3270 server (IBM S/3XX mainframes) is True. Default value for telnet 5250 server (IBM AS/400 mainframes) is **False**.

### 4.1.2.1.2.16 LastErrMsg

Returns a string that specifies the last communication error reported by the host in the telnet current connection.

**Visual Basic Syntax**

```vbnet
[String =] TNBnnnnX.LastErrMsg
```

**Delphi Syntax**

```delphi
[String :=] TNBnnnnX.LastErrMsg;
```

### 4.1.2.1.2.17 Password

Sets a Password for the UserId specified.

**Visual Basic Syntax**

```vbnet

```
**Password**

*Delphi Syntax*

```delphi
TNBnnnxn. Password := String;
```

See also **DeviceName** and **UserId** properties.

### 4.1.2.1.2.18 Profiles

Sets the **TNBXProfiles** ActiveX control as the profile manager for this control.

**Visual Basic Syntax**

```vbnet
TNBXEmulator. ProfilesControl := TNBXProfiles
```

**Delphi Syntax**

```delphi
TNBXEmulator. ProfilesControl := TNBXProfiles;
```

**Remarks**

After setting this property, you can access to a collection of connections to be generated and customized through the **ShowEditor** method.

See Also **TNBXProfiles** control, **Subkey** property and **ShowEditor** method.

### 4.1.2.1.2.19 Port

TCP Port of the TN3270/TN5250 host to connect to.

**Visual Basic Syntax**

```vbnet
TNBnnnnx. Port := Integer
```

**Delphi Syntax**

```delphi
TNBnnnnx. Port := Integer;
```

**Remarks**

By default it takes the standard telnet port. You can change it if the non standard port is to be used.

Default value is 23.
See also `Host` property, `Connect` method and `OnConnect` event.

### 4.1.2.1.2.20 Rows

Return the total number of rows of the current terminal type.

**Visual Basic Syntax**

```vbnet
[Integer =] TNBnnnnX.Rows
```

**Delphi Syntax**

```delphi
[Integer :=] TNBnnnnX.Rows;
```

**Remarks**

Depending on the terminal type chosen, this property returns the values 24, 32 or 43.

See Also `Cols` and `TerminalType` property.

### 4.1.2.1.2.21 ScreenRow

Returns a string contained in the specified row.

**Visual Basic Syntax**

```vbnet
[String =] TNBnnnnX.ScreenRow (Row As Integer)
```

**Delphi Syntax**

```delphi
[String :=] TNBnnnnX.ScreenRow [Row:Integer];
```

**Remarks**

If you specified a wrong row value, a range error will occur.

See Also `HostFields` property.

### 4.1.2.1.2.22 SessionId

Returns a string containing a unique identifier for the telnet object.

**Visual Basic Syntax**

```vbnet
[String =] TNBnnnnX.SessionId
```

See Also `TNBXPOOL` class.
4.1.2.1.2.23 SessionState

Returns the current session state.

**Visual Basic Syntax**

```vbnet
[TNBXSS] = TNBnnnnX.SessionState
```

**Delphi Syntax**

```delphi
[TNBXSS] := TNBnnnnX.SessionState;
```

It can return any of TNBSessionState constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssNoSession</td>
<td>Not in session.</td>
</tr>
<tr>
<td>ssSSCPLU</td>
<td>In session with VTAM.</td>
</tr>
<tr>
<td>ssLULU</td>
<td>In session with VTAM Application.</td>
</tr>
</tbody>
</table>

**Remarks**

While the telnet connection is closed this property returns always ssNoSession value. When the telnet connection is opened it can return:

- ssNoSession, indicating that the connection was established at telnet level but not at 5250 Data Stream level.
- ssSSCPLU, indicating that a connection with the VTAM was established, but there's not a session with an final application (like CICS, TSO, etc.). (This state is only valid for TN3270 Hosts).
- ssLULU, indicating that the connection with the VTAM application (like CICS, TSO, etc.) has been established.

See Also `TNBXSS` type.

4.1.2.1.2.24 SSL

Allows to enable the SSL protocol (Secure Sockets Layer).

**Visual Basic Syntax**

```vbnet
TNBnnnnX.SSL [= Boolean]
```

**Delphi Syntax**

```delphi
TNBnnnnX.SSL := Boolean;
```

See also `SSLMethod`, `SSLDelay`, `SSLCertFile` properties.

4.1.2.1.2.25 SSLAcceptExpired

Allows to accept or not expired certificates.
**Visual Basic Syntax**

`TNBnXnX.X.SSLAcceptExpired [ = Boolean]`

**Delphi Syntax**

`TNBnXnX.X.SSLAcceptExpired [:= Boolean];`

See also [SSL], [SSLAcceptInvalid], [SSLAcceptInvalidCA], [SSLAcceptNotYetValid], [SSLAcceptSelfSigned] properties.

---

### 4.1.2.1.2.26 SSLAcceptInvalid

Allows to accept or not any invalid certificate.

**Visual Basic Syntax**

`TNBnXnX.X.SSLAcceptInvalid [ = Boolean]`

**Delphi Syntax**

`TNBnXnX.X.SSLAcceptInvalid [:= Boolean];`

See also [SSL], [SSLAcceptExpired], [SSLAcceptInvalidCA], [SSLAcceptNotYetValid], [SSLAcceptSelfSigned] properties.

---

### 4.1.2.1.2.27 SSLAcceptInvalidCA

Allows to accept or not any invalid CA certificate.

**Visual Basic Syntax**

`TNBnXnX.X.SSLAcceptInvalidCA [ = Boolean]`

**Delphi Syntax**

`TNBnXnX.X.SSLAcceptInvalidCA [:= Boolean];`

See also [SSL], [SSLAcceptExpired], [SSLAcceptInvalid], [SSLAcceptNotYetValid], [SSLAcceptSelfSigned] properties.

---

### 4.1.2.1.2.28 SSLAcceptNotYetValid

Allows to accept or not certificates not yet valid.

**Visual Basic Syntax**
TNBnnnnX.SSLAcceptNotYetValid [= Boolean]

**Delphi Syntax**

`TNBnnnnX.SSLAcceptNotYetValid [= Boolean];`

See also SSL, SSLAcceptExpired, SSLAcceptInvalid, SSLAcceptInvalidCA, SSLAcceptSelfSigned properties.

**4.1.2.1.2.29 SSLAcceptSelfSigned**

Allows to accept or not self signed certificates.

**Visual Basic Syntax**

`TNBnnnnX.SSLAcceptSelfSigned [= Boolean]`

**Delphi Syntax**

`TNBnnnnX.SSLAcceptSelfSigned [= Boolean];`

See also SSL, SSLAcceptExpired, SSLAcceptInvalid, SSLAcceptInvalidCA, SSLAcceptSelfSigned properties.

**4.1.2.1.2.30 SSLCertFile**

Sets/gets the SSL Certificate file.

**Visual Basic Syntax**

`TNBnnnnX.SSLCertFile [= String]`

**Delphi Syntax**

`TNBnnnnX.SSLCertFile [= String];`

See also SSL, SSLMethod, SSLCertPassword properties.

**4.1.2.1.2.31 SSLCertPassword**

Sets/gets the SSL Certificate password.

**Visual Basic Syntax**

`TNBnnnnX.SSLCertPassword [= String]`

**Delphi Syntax**

`TNBnnnnX.SSLCertPassword [= String];`
See also SSL, SSLMethod, SSLCertFile properties.

4.1.2.1.2.32 SSLDisplay
Enable/disable the display certificate mode.

**Visual Basic Syntax**

```vbnet
TNBnnnnX.SSLDisplay [= Boolean]
```

**Delphi Syntax**

```delphi
TNBnnnnX.SSLDisplay := Boolean;
```

See also SSL property.

4.1.2.1.2.33 SSLEnableDialogs
Allows to enable SSL dialogs mechanism.

**Visual Basic Syntax**

```vbnet
TNBnnnnX.SSLEnableDialogs [= Boolean]
```

**Delphi Syntax**

```delphi
TNBnnnnX.SSLEnableDialogs := Boolean;
```

4.1.2.1.2.34 SSLKeyFile
Sets/gets the SSL key file.

**Visual Basic Syntax**

```vbnet
TNBnnnnX.SSLKeyFile [= String]
```

**Delphi Syntax**

```delphi
TNBnnnnX.SSLKeyFile := String;
```

See also SSL, SSLMethod, SSLCertPassword, SSLCertFile and SSLKeyFile properties.

4.1.2.1.2.35 SSLMethod
Sets/gets the supported SSL method.

**Visual Basic Syntax**

```vbnet
TNBnnnnX.SSLMethod [= TNBSSLMethod]
```

**Delphi Syntax**
TNBnnnnX. SSLMethod [:= TNBSSLMethod];

See also SSL, SSLCertPassword, SSLCertFile properties and TNBSSLMethod constants.

4.1.2.1.2.36 SSLRootCertFile
Sets/gets the SSL root's certification file.

**Visual Basic Syntax**

TNBnnnnX. SSLRootCertFile [= String]

**Delphi Syntax**

TNBnnnnX. SSLRootCertFile [:= String];

See also SSL, SSLCertPassword, SSLMethod and SSLCertFile.

4.1.2.1.2.37 SubKey
Sets/gets the subkey under which will be stored the connection profiles.

**Visual Basic Syntax**

TNBnnnnX. SubKey [= String]

**Delphi Syntax**

TNBnnnnX. SubKey [:= String];

**Remarks**

This property is only valid if the ProfilesControl property has been assigned to a TNBXProfiles control.

Default value is "TNB3270E" for TNB3270X control and "TNB5250" for TNB5250X control.

See also ProfilesControl property and TNBXProfiles Control.

4.1.2.1.2.38 SynchronizeEvents
Controls whether synchronization of events with a main window thread is required or not.

**Visual Basic Syntax**

TNBnnnnX. SynchronizeEvents [= Boolean]

**Delphi Syntax**

TNBnnnnX. SynchronizeEvents [:= Boolean];
Remarks

In windowed synchronization of events with the main window thread is required to prevent corruption in the gui framework. In non-windowed applications, this property should be set to false, to avoid loss of events.

Default value is True.

4.1.2.1.2.39 TerminalType

Sets the terminal type.

Visual Basic Syntax

TNBnnnnX.TerminalType [= TNBX5250TT] TerminalModel ???

Delphi Syntax

TNBnnnnX.TerminalType [= TNBX5250TT];

It can take any of the following constant values:

For 5250:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt5211m2 = 0</td>
<td>IBM-5211-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3179m2 = 1</td>
<td>IBM-3179-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3477mFC = 2</td>
<td>IBM-3477-FC 27 rows x 132 columns</td>
</tr>
<tr>
<td>tt3180m2 = 3</td>
<td>IBM-3180-2 27 rows x 132 columns</td>
</tr>
</tbody>
</table>

Visual Basic Syntax

TNBnnnnX.TerminalType [= TNBX3270TT]

Delphi Syntax

TNBnnnnX.TerminalType [= TNBX3270TT];

For 3270:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt3278m2 = 0</td>
<td>IBM-3278-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m2E = 1</td>
<td>IBM-3278-2-E 24 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m3 = 2</td>
<td>IBM-3278-3 32 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m3E = 3</td>
<td>IBM-3278-3-E 32 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m4 = 4</td>
<td>IBM-3278-4 43 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m4E = 5</td>
<td>IBM-3278-4-E 43 rows x 80 columns extended</td>
</tr>
</tbody>
</table>

See also TNBX5250TT and TNBX3270TT constants.
4.1.2.1.2.40  UserId

Sets a UserId for the device name specified.

**Visual Basic Syntax**

TNBnnnnX. UserId [ = String ]

**Delphi Syntax**

TNBnnnnX. UserId [:= String ];

See also **DeviceName** and **Password** properties.

4.1.2.1.2.41  XLockDelay

Controls the delay in milliseconds between the last **OnScreenChange** and **OnSystemUnlock** events.

**Visual Basic Syntax**

TNBnnnnX. XLockDelay [ = Integer ]

**Delphi Syntax**

TNBnnnnX. XLockDelay [ = Integer ];

4.1.2.1.3  Methods

4.1.2.1.3.1  AsVariant

Returns the object as variant variable type.

**Visual Basic Syntax**

[ Variant = ] TNBnnnnX. AsVariant ()

**Delphi Syntax**

[ Variant := ] TNBnnnnX. AsVariant ();

4.1.2.1.3.2  ClassNameIs

Indicates if a specified name is the corresponding class name.

**Visual Basic Syntax**

[ Boolean = ] TNBnnnnX. ClassNameIs (Name As String)

**Delphi Syntax**

[ Boolean := ] TNBnnnnX. ClassNameIs (Name: string);
4.1.2.1.3.3 CloseEditor

Closes the modal dialog with the connection properties.

**Visual Basic Syntax**

\[TNBnnnnX.CloseEditor \text{(Apply As Boolean)}\]

**Delphi Syntax**

\[TNBnnnnX.CloseEditor \text{(Apply:boolean);}\]

See also ProfilesControl property, TNBXProfiles control and ShowEditor method.

4.1.2.1.3.4 Connect

The Connect method establishes the connection to the Telnet server. The Host and Port properties must be assigned for the connection to be successfully established.

**Visual Basic Syntax**

\[TNBnnnnX.Connect\]

**Delphi Syntax**

\[TNBnnnnX.Connect;\]

**Remarks**

If the specified host doesn't exist or if the connection fails, the OnConnectionFail event is fired. If the connection is successfully established, the OnConnect event is fired.

See also Host and Port properties, Disconnect method, OnConnect and OnConnectionFail events.

4.1.2.1.3.5 Disconnect

The Disconnect method ends the connection with the Telnet server.

**Visual Basic Syntax**

\[TNBnnnnX.Disconnect\]

**Delphi Syntax**

\[TNBnnnnX.Disconnect;\]
Remarks

After the client disconnects, the OnDisconnect event is fired. If the client is not connected to any server, the disconnect method has no effect.

See also Connect method and OnDisconnect event.

4.1.2.1.3.6 GetScreenRowEx

GetScreenRowEx can be used to retrieve the text buffer of the one row of the current screen. Additionally, you can retrieve the standard or extended attribute corresponding to each field.

Visual Basic Syntax

[String =] TNBnnnnX.GetScreenRowEx (Row As Integer, [Attr As Boolean = True], [Eab As Boolean = True])

Delphi Syntax

[String :=] TNBnnnnX.GetScreenRowEx (Row:Integer, [Attr:boolean = True], [Eab:boolean = True]);

Remarks

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in Row. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for fields attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

See also GetScreenText method.

4.1.2.1.3.7 GetScreenText

GetScreenText returns the text buffer of a portion of the current screen. You can choose to recover the attribute or extended attribute of each field together to the character data.

Visual Basic Syntax

[String =] TNBnnnnX.GetScreenText (StartPos As Integer, EndPos As Integer, [Attr As Boolean = True], [Eab As Boolean = True])

Delphi Syntax
[String :=] TNBnnnnX.GetScreenText (StartPos:integer, EndPos:integer, [Attr:boolean = True], [Eab:boolean = True]);

**Remarks**

StartPos and EndPos specify the start position and the end position of the text buffer that will be returned.

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in Row. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for fields attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

See also GetScreenRowEx method.

**4.1.2.1.3.8 IsConnected**

Returns a boolean value indicating if a connection is currently established with the Host.

**Visual Basic Syntax**

[Boolean :=] TNBnnnnX.IsConnected

**Delphi Syntax**

[Boolean :=] TNBnnnnX.IsConnected;

See also Connect and Disconnect methods.

**4.1.2.1.3.9 LoadFromXMLFile**

Loads the XML code corresponding to the TNBnnnnX object.

**Visual Basic Syntax**

TNBnnnnX.LoadFromXMLFile (XMLFile As String)

**Delphi Syntax**

TNBnnnnX.LoadFromXMLFile (XMLFile:string);

See also SaveToXMLFile method.
4.1.2.1.3.10 SaveToXMLFile

Exports the XML code corresponding to the TNBnnnnX object.

**Visual Basic Syntax**

TNBnnnnX.SaveToXMLFile (XMLFile As String)

**Delphi Syntax**

TNBnnnnX.SaveToXMLFile (XMLFile: string);

See also LoadFromXMLFile method.

4.1.2.1.3.11 SendAid

Sets an Attention Identifier Key and sends the modified fields to the Mainframe. Modified fields can be input fields (unprotected) or sometimes protected fields, having the property Modified set to True.

**Visual Basic Syntax**

TNBnnnnX.SendAid (AidKey As string)

**Delphi Syntax**

TNBnnnnX.SendAid (AidKey:string);

This method works like the AidKey property, but it takes a string with the name of the AID key as a parameter. You can specify only one key for each call to the method.

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
</tbody>
</table>
PF12  PF12 key
PF13  PF13 key
PF14  PF14 key
PF15  PF15 key
PF16  PF16 key
PF17  PF17 key
PF18  PF18 key
PF19  PF19 key
PF20  PF20 key
PF21  PF21 key
PF22  PF22 key
PF23  PF23 key
PF24  PF24 key
PA1   PA1 key

See also **AIDKey** property.

4.1.2.1.3.12 SendKeys

SendKeys can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host functions.

**Visual Basic Syntax**

TNBnnnnX.SendKeys (DataString As String)

**Delphi Syntax**

TNBnnnnX.SendKeys (DataString:string);

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
</tbody>
</table>
Here is an example of the Keys parameter:
"logon applid(cics)@E"

**Remarks**

The execution of the SendKeys is stopped when an AID Key is sent to the Host.

See also **AIDKey** property.

4.1.2.1.3.13 ShowEditor

Shows a modal dialog with the connection properties.

**Visual Basic Syntax**

`TNBnnnnX.ShowEditor`

**Delphi Syntax**

`TNBnnnnX.ShowEditor;`

**Remarks**

If you haven't assigned the ProfilesControl property, the connection profiles list will not be shown. Setting the ProfilesControl property to a valid `TNBXProfiles`, allows you to define connection profiles at run time and save them into a file for future reuse.

See also **ProfilesControl** property and **TNBXProfiles** control.
4.1.2.1.4 Events

4.1.2.1.4.1 OnConnect

Occurs after a successfully connection to the server is established.

See also Connect method and, OnConnectionFail and OnDisconnect events.

4.1.2.1.4.2 OnConnectionFail

Occurs after the connection to the server fails.

See also Connect method, OnConnect and OnDisconnect events.

4.1.2.1.4.3 OnDisconnect

Occurs after a disconnection of the server.

See also Disconnect method.

4.1.2.1.4.4 OnMessageWaitingOff

Occurs when the Message Waiting status of the status bar turns to off.

See also OnMessageWaitingOn event.

4.1.2.1.4.5 OnMessageWaitingOn

Occurs when the Message Waiting status of the status bar turns to on.

See also OnMessageWaitingOff event.

4.1.2.1.4.6 OnScreenChange

Occurs after a new data screen has arrived.

See also OnSystemLock and OnSystemUnlock events.

4.1.2.1.4.7 OnSendAid

Occurs before an Aid key is to be sent.

Remarks

This event is fired when a SendAid or SendKeys methods or the AidKey property are used to send an Aid Key and data to the mainframe. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator control.

See also SendAid, SendKeys methods and AidKey property from telnet controls, and Lock method from TNBXFields Class.

4.1.2.1.4.8 OnSessionState

Occurs when a session-state change takes place.

Remarks
A session state can be ssNoSession, ssSSCPLU and ssLULU. Any transition between the states fires this event.

See also **SessionState** property.

### 4.1.2.1.4.9 OnSystemLock

Occurs when a terminal changes to a system locked state.

**Remarks**

During this state, the terminal is waiting for any response from the mainframe. Sending data isn't possible until the **OnSystemUnlock** event is fired.

See also **OnSystemUnlock** event.

### 4.1.2.1.4.10 OnSystemUnlock

Occurs when a terminal changes to a system unlocked state.

**Remarks**

Only during this state, the component can send data to the host system.

See also **OnSystemLock** and **OnScreenChange** events.

### 4.1.2.2 Tnb3287X Reference

#### 4.1.2.2.1 Tnb3287X Control

**Properties**

- **AttachMode**
- **CodePage**
- **ConversionTable**
- **Debug**
- **DebugDir**
- **DeviceName**
- **Host**
- **IsConnected**
- **Port**
- **PrinterCfg**
- **ProfilesControl**
- **SendToPrinter**
- **SubKey**
- **Text**

**Methods**

- **AboutBox**
- **Connect**
- **Disconnect**
- **ShowPrinterEditor**

**Events**
4.1.2.2.1.1 Properties

Controls the type of association between the emulation session and the printing session.

**Visual Basic Syntax**

\[ Tnb3287X.AttachMode = TnbAttachMode \]

**Delphi Syntax**

\[ Tnb3287X.AttachMode := TnbAttachMode \]

**Remarks**

The following are possible values for AttachMode property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>amDirect</td>
<td>Enables direct association between sessions.</td>
</tr>
<tr>
<td>amAssociate</td>
<td>The host handle the association between sessions.</td>
</tr>
</tbody>
</table>

Default value is **amDirect**.

Specifies the internal ASCII-EBCDIC conversion table.

**Visual Basic Syntax**

\[ Tnb3287X.CodePage = TNBXCP \]

**Delphi Syntax**

\[ Tnb3287X.CodePage := TNBXCP \]

\( TNBXCP \) can take any of the following Code Pages constant values:
### Constant Value

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia = 037</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium = 037</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada = 037</td>
<td>Canada</td>
</tr>
<tr>
<td>cpCzechRepublic = 870</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cpDenmark = 277</td>
<td>Denmark</td>
</tr>
<tr>
<td>cpFinland = 278</td>
<td>Finland</td>
</tr>
<tr>
<td>cpFrance = 297</td>
<td>France</td>
</tr>
<tr>
<td>cpGermany = 273</td>
<td>Germany</td>
</tr>
<tr>
<td>cpGreece = 423</td>
<td>Greece</td>
</tr>
<tr>
<td>cpGreece2 = 875</td>
<td>Greece</td>
</tr>
<tr>
<td>cpItaly = 280</td>
<td>Italy</td>
</tr>
<tr>
<td>cpLatinAmerica = 284</td>
<td>Latin America</td>
</tr>
<tr>
<td>cpNetherlands = 037</td>
<td>Netherlands</td>
</tr>
<tr>
<td>cpNorway = 277</td>
<td>Norway</td>
</tr>
<tr>
<td>cpPoland = 870</td>
<td>Poland</td>
</tr>
<tr>
<td>cpPortugal = 037</td>
<td>Portugal</td>
</tr>
<tr>
<td>cpSpain = 284</td>
<td>Spain</td>
</tr>
<tr>
<td>spSweden = 278</td>
<td>Sweden</td>
</tr>
<tr>
<td>cpSwitzerlandFrench = 500</td>
<td>Switzerland (French)</td>
</tr>
<tr>
<td>cpSwitzerlandGerman = 500</td>
<td>Switzerland (German)</td>
</tr>
<tr>
<td>cpUnitedKingdom = 285</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>cpUnitedStates = 037</td>
<td>United States</td>
</tr>
<tr>
<td>cpInternational = 500</td>
<td>International</td>
</tr>
</tbody>
</table>

Default **cpUnitedStates**.

See also **ConversionTable** property and Code Pages constants.

Specifies an external ASCII-EBCDIC conversion table to be loaded from a file. If no one is specified, an internal conversion table is used, according to **CodePage** property setting.

**Visual Basic Syntax**

```vbnet
Tnb3287X.ConversionTable [= String]
```

**Delphi Syntax**

```delphi
Tnb3287X.ConversionTable; [:= String]
```

**Remarks**
De specified file must exist and contain a valid conversion table format.

The format of conversion table is:

[Ascii-To-Ebcdic]
Ascii hexadecimal code 1 = Ebcdic hexadecimal code 1
Ascii hexadecimal code 2 = Ebcdic hexadecimal code 2
... ...
Ascii hexadecimal code n = Ebcdic hexadecimal code n

[Ebcdic-To-Ascii]
Ebcdic hexadecimal code 1 = Ascii hexadecimal code 1
Ebcdic hexadecimal code 2 = Ascii hexadecimal code 2
... ...
Ebcdic hexadecimal code n = Ascii hexadecimal code n

See also CodePage property.

Allows the generation of trace information for debugging purposes. The debugging can help you to track down both, errors and logic information.

**Visual Basic Syntax**

```
Tnb3287X.Debug [= Boolean]
```

**Delphi Syntax**

```
Tnb3287X.Debug; [:= Boolean]
```

**Remarks**

The following are possible values for Debug property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Enables debug information.</td>
</tr>
<tr>
<td>False</td>
<td>Disables debug information.</td>
</tr>
</tbody>
</table>

Default value is **False**.

By default, the debug file is saved in the application directory. You can change the directory by setting the **DebugDir** property.

See also **DebugDir** property.
Specifies the directory for debug files.

**Visual Basic Syntax**

```vbnet
Tnb3287X.DebugDir [ = String]
```

**Delphi Syntax**

```delphi
Tnb3287X.DebugDir; [: = String]
```

**Remarks**

By default, the debug file is saved in the application's directory. You can change it by setting this property to another directory. This directory must exist, otherwise, no debug information will be generated and an error will not occur. The default value is the application directory.

See also [Debug](#) property.

Sets an specific telnet device name defined in the telnet server.

**Visual Basic Syntax**

```vbnet
Tnb3287X.DeviceName [ = String]
```

**Delphi Syntax**

```delphi
Tnb3287X.DeviceName; [: = String]
```

**Remarks**

If no name is given, the telnet server will assign an available device from a public pool in case that it exists.

See also [Host](#) property.
TCP/IP address or DNS name of the Host to connect to. It can be a name like 'myhost.mydomain.com' or directly an IP address such as "206.155.164.20".

**Visual Basic Syntax**

\[ Tnb3287X.Host [= String] \]

**Delphi Syntax**

\[ Tnb3287X.Host; [= String] \]

**Remarks**

The host name stored in the Host property must be alpha numeric, and can't contain spaces nor special characters (i.e. : " / ? \)

See also Port property, Connect method and OnConnect event.

Returns a boolean value indicating if a connection is currently established with the Host.

**Visual Basic Syntax**

\[ Boolean =] Tnb3287X.IsConnected \]

**Delphi Syntax**

\[ Boolean :=] Tnb3287X.IsConnected; \]

See also Connect and Disconnect methods.

Sets/gets the TCP Port of the TN3270/TN5250 host to connect to.

**Visual Basic Syntax**

\[ Tnb3287X.Port [= Integer] \]

**Delphi Syntax**

\[ Tnb3287X.Port; [= Integer] \]
Remarks

By default it takes the standard telnet port. You can change it if the non standard port is to be used. Default value is 23.

See also Host property, Connect method and OnConnect event.

Points to the TnbXPrinter object which control the printer activity.

Visual Basic Syntax

[TnbXPrinter =] Tnb3287X.PrinterCfg

Delphi Syntax

[TnbXPrinter :=] Tnb3287X.PrinterCfg;

Sets the ProfilesControl control as the profile manager for this control.

Visual Basic Syntax

Tnb3287X.ProfilesControl [ = TnbXProfiles]

Delphi Syntax

Tnb3287X.ProfilesControl; [ := TnbXProfiles]

See Also Subkey property.

Controls the capability of sending a document to a printer.

Visual Basic Syntax

Tnb3287X.SendToPrinter [ = Boolean]

Delphi Syntax
Sets/gets the subkey under which it will be stored the connection profiles.

**Visual Basic Syntax**

\[ Tnb3287X.SendToPrinter; [ := \text{Boolean} ] \]

**Delphi Syntax**

\[ Tnb3287X.SendToPrinter; [ := \text{Boolean} ] \]

**Remarks**

This property is only valid if the ProfilesControl property is being used.

See also ProfilesControl Control.

Returns the received printer-data.

**Visual Basic Syntax**

\[ \text{String} \ = \ Tnb3287X.Text \]

**Delphi Syntax**

\[ \text{String} \ := \ Tnb3287X.Text; \]

**Remarks**

When a OnDataAvailable event is fired, Text property returns the text data just received. Once the OnEndDocument event is fired, the Text property holds the whole received printer-data.
4.1.2.1.2 Methods

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```
TTnb3287X.AboutBox
```

**Delphi Syntax**

```
TTnb3287X.AboutBox;
```

The Connect method establishes the connection to the Telnet server. The **Host** and **Port** properties must be assigned for the connection to be successfully established.

**Visual Basic Syntax**

```
Tnb3287X.Connect
```

**Delphi Syntax**

```
Tnb3287X.Connect;
```

**Remarks**

If the specified host doesn’t exist or if the connection fails, the **OnConnectionFail** event is fired. If the connection is successfully established, the **OnConnect** event is fired.

*See also* **Host** and **Port** properties, **Disconnect** method, **OnConnect** and **OnConnectionFail** events.

The Disconnect method ends the connection with the Telnet server.

**Visual Basic Syntax**

```
Tnb3287X.Disconnect
```

**Delphi Syntax**

```
Tnb3287X.Disconnect;
```
Remarks

After the client disconnects, the OnDisconnect event is fired. If the client is not connected to any server, the disconnect method has no effect.

See also Connect method and OnDisconnect event.

Shows the printer’s configuration dialog.

Visual Basic Syntax

Tnb3287X.ShowPrinterEditor

Delphi Syntax

Tnb3287X.ShowPrinterEditor;

4.1.2.1.3 Events

Occurs after a successfully connection to the server is established.

See also Connect method and, OnConnectionFail and OnDisconnect events.

Occurs after the connection to the server fails.

See also Connect method, OnConnect and OnDisconnect events.

Occurs after a data file buffer is received.

See also Text property, OnStartDocument and OnEndDocument events.
Occurs after a disconnection of the server.

See also **Disconnect** method.

Occurs after the last data file buffer is received.

See also **OnStartDocument** event.

Occurs after the first data file buffer is received.

See also **OnEndDocument** event.

4.1.2.3  Tnb3812X Reference

4.1.2.3.1  Tnb3812X Control

**Properties**

- CodePage
- ConversionTable
- Debug
- DebugDir
- DeviceName
- Host
- IsConnected
- MsgQueueLib
- MsgQueueName
- Port
- PrinterCfg
- ProfilesControl
- SendToPrinter
- SubKey
- Text

**Methods**

- AboutBox
4.1.2.3.1.1 Properties

Specifies the internal ASCII-EBCDIC conversion table.

**Visual Basic Syntax**

\[ Tnb3812X.CodePage \] = TnbXCP

**Delphi Syntax**

\[ Tnb3812X.CodePage; \] := TnbXCP

TNBXCP can take any of the following Code Pages constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia = 037</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium = 037</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada = 037</td>
<td>Canada</td>
</tr>
<tr>
<td>cpCzechRepublic = 870</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cpDenmark = 277</td>
<td>Denmark</td>
</tr>
<tr>
<td>cpFinland = 278</td>
<td>Finland</td>
</tr>
<tr>
<td>cpFrance = 297</td>
<td>France</td>
</tr>
<tr>
<td>cpGermany = 273</td>
<td>Germany</td>
</tr>
<tr>
<td>cpGreece = 423</td>
<td>Greece</td>
</tr>
<tr>
<td>cpGreece2 = 875</td>
<td>Greece</td>
</tr>
<tr>
<td>cpItaly = 280</td>
<td>Italy</td>
</tr>
<tr>
<td>cpLatinAmerica = 284</td>
<td>Latin America</td>
</tr>
<tr>
<td>cpNetherlands = 037</td>
<td>Netherlands</td>
</tr>
<tr>
<td>cpNorway = 277</td>
<td>Norway</td>
</tr>
<tr>
<td>cpPoland = 870</td>
<td>Poland</td>
</tr>
<tr>
<td>cpPortugal = 037</td>
<td>Portugal</td>
</tr>
<tr>
<td>cpSpain = 284</td>
<td>Spain</td>
</tr>
</tbody>
</table>
Specifies an external ASCII-EBCDIC conversion table to be loaded from a file. If no one is specified, an internal conversion table is used, according to `CodePage` property setting.

**Visual Basic Syntax**

```
Tnb3812X.ConversionTable [= String]
```

**Delphi Syntax**

```
Tnb3812X.ConversionTable; [:= String]
```

**Remarks**

The specified file must exist and contain a valid conversion table format.

The format of conversion table is:

- `[Ascii-To-Ebcdic]`  
  Ascii hexadecimal code 1 = Ebcdic hexadecimal code 1  
  Ascii hexadecimal code 2 = Ebcdic hexadecimal code 2  
  ...  
  Ascii hexadecimal code n = Ebcdic hexadecimal code n

- `[Ebcdic-To-Ascii]`  
  Ebcdic hexadecimal code 1 = Ascii hexadecimal code 1  
  Ebcdic hexadecimal code 2 = Ascii hexadecimal code 2  
  ...  
  Ebcdic hexadecimal code n = Ascii hexadecimal code n

See also `CodePage` property.
Allows the generation of trace information for debugging purposes. The debugging can help you to track down both, errors and logic information.

**Visual Basic Syntax**

```vbnet
Tnb3812X.Debug [= Boolean]
```

**Delphi Syntax**

```delphi
Tnb3812X.Debug; [: = Boolean]
```

**Remarks**

The following are possible values for Debug property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Enables debug information.</td>
</tr>
<tr>
<td>False</td>
<td>Disables debug information.</td>
</tr>
</tbody>
</table>

Default value is **False**.

By default, the debug file is saved in the application directory. You can change the directory by setting the `DebugDir` property.

**See also** `DebugDir` property.

Specifies the directory for debug files.

**Visual Basic Syntax**

```vbnet
Tnb3812X.DebugDir [= String]
```

**Delphi Syntax**

```delphi
Tnb3812X.DebugDir; [: = String]
```

**Remarks**

By default, the debug file is saved in the application's directory. You can change it by setting this property to another directory. This directory must exist, otherwise, no debug information will be generated and an error will not occur.
The default value is the application directory.

See also **Debug** property.

Sets/gets an specific telnet device name defined in the telnet server.

**Visual Basic Syntax**

```
Tnb3812X.DeviceName [= String]
```

**Delphi Syntax**

```
Tnb3812X.DeviceName := String
```

**Remarks**

If no name is given, the telnet server will assign an available device from a public pool in case of it exists.

See also **Host** property.

Sets/gets the TCP/IP address or DNS name of the Host to connect to. It can be a name like 'myhost.mydomain.com' or directly an IP address such as "206.155.164.20".

**Visual Basic Syntax**

```
Tnb3812X.Host [= String]
```

**Delphi Syntax**

```
Tnb3812X.Host; [:= String]
```

**Remarks**

The host name stored in the Host property must be alpha numeric, and can't contain spaces nor special characters (i.e. : ": / ? \"

See also **Port** property, **Connect** method and **OnConnect** event.
Returns a boolean value indicating if a connection is currently established with the Host.

**Visual Basic Syntax**

```vbnet
[Boolean =] Tnb3812X.IsConnected
```

**Delphi Syntax**

```delphi
[Boolean :=] Tnb3812X.IsConnected;
```

See also [Connect](#) and [Disconnect](#) methods.

Holds the library that contains the message used by the print writer for operational messages. The default value is "*LIBL".

**Visual Basic Syntax**

```vbnet
Tnb3812X.MsgQueueLib [= String]
```

**Delphi Syntax**

```delphi
Tnb3812X.MsgQueueLib; [= String]
```

See also [MsgQueueName](#) property.

Holds the name of the message queue used by the print writer for operational messages. The default value is "QSYSOPR".

**Visual Basic Syntax**

```vbnet
Tnb3812X.MsgQueueName [= String]
```

**Delphi Syntax**

```delphi
Tnb3812X.MsgQueueName; [= String]
```
Sets/gets the TCP Port of the TN3270/TN5250 host to connect to.

**Visual Basic Syntax**

\[ Tnb3812X.Port \[= Integer] \]

**Delphi Syntax**

\[ Tnb3812X.Port; [= Integer] \]

**Remarks**

By default it takes the standard telnet port. You can change it if the non standard port is to be used.

Default value is 23.

See also **Host** property, **Connect** method and **OnConnect** event.

Points to the TnbXPrinter object that extends the TPrinter standard Delphi object to control the printer activity.

**Visual Basic Syntax**

\[ TnbXPrinter = Tnb3812X.PrinterCfg \]

**Delphi Syntax**

\[ TnbXPrinter := Tnb3812X.PrinterCfg; \]

See also **SendToPrinter** property.

Sets the TnbXProfiles control as the profile manager for this control.
**Visual Basic Syntax**

*Tnb3812X.ProfilesControl [ = TnbXProfiles]*

**Delphi Syntax**

*Tnb3812X.ProfilesControl; [: = TnbXProfiles]*

See Also **Subkey** property.

Controls the capability of send a document to a printer.

**Visual Basic Syntax**

*Tnb3812X.SendToPrinter [ = Boolean]*

**Delphi Syntax**

*Tnb3812X.SendToPrinter; [: = Boolean]*

See also **Printer** property.

Sets/gets the subkey under which will be stored the connection profiles.

**Visual Basic Syntax**

*Tnb3812X.SubKey [ = String]*

**Delphi Syntax**

*Tnb3812X.SubKey; [: = String]*

**Remarks**

This property is only valid if the **TNBProfiles** property is being used.

See also **TNBProfiles** Control.
Returns the received printer-data.

**Visual Basic Syntax**

[String =] Tnb3812X.Text

**Delphi Syntax**

[String :=] Tnb3812X.Text;

**Remarks**

When a **OnDataAvailable** event is fired, Text property returns the text data just received. Once the **OnEndDocument** event is fired, the Text property holds the whole received printer-data.

### 4.1.2.3.1.2 Methods

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

Tnb3812X.AboutBox

**Delphi Syntax**

Tnb3812X.AboutBox;

The Connect method establishes the connection to the Telnet server. The **Host** and **Port** properties must be assigned for the connection to be successfully established.

**Visual Basic Syntax**

Tnb3812X.Connect

**Delphi Syntax**

Tnb3812X.Connect;
Remarks

If the specified host doesn't exist or if the connection fails, the OnConnectionFail event is fired. If the connection is successfully established, the OnConnect event is fired.

See also Host and Port properties, Disconnect method, OnConnect and OnConnectionFail events.

The Disconnect method ends the connection with the Telnet server.

Visual Basic Syntax

Tnb3812X.Disconnect

Delphi Syntax

Tnb3812X.Disconnect;

Remarks

After the client disconnects, the OnDisconnect event is fired. If the client is not connected to any server, the disconnect method has no effect.

See also Connect method and OnDisconnect event.

Shows the printer's configuration dialog.

Visual Basic Syntax

Tnb3812X.ShowPrinterEditor

Delphi Syntax

Tnb3812X.ShowPrinterEditor;
4.1.2.3.1.3 Events

Occurs after a successfully connection to the server is established.

See also **Connect** method and, **OnConnectionFail** and **OnDisconnect** events.

Occurs after the connection to the server fails.

See also **Connect** method, **OnConnect** and **OnDisconnect** events.

Occurs after a data file buffer is received.

See also **Text** property, **OnStartDocument** and **OnEndDocument** events.

Occurs after a disconnection of the server.

See also **Disconnect** method.

Occurs after the last data file buffer is received.

See also **OnStartDocument** event.

Occurs after the first data file buffer is received.

See also **OnEndDocument** event.
4.1.2.4 TNBXPOOL Reference

4.1.2.4.1 TNBXPOOL Class

This object is used for administering pools of telnet objects.

**Properties**

- **Id**
- **Type**

**Methods**

- **Acquire**
- **Release**

4.1.2.4.2 Properties

4.1.2.4.2.1 AcquiredCount

Returns the amount of allocated objects that are in use.

**Visual Basic Syntax**

```
[Integer  =]  TNBXPOOL.Count
```

See also **Count** and **ReleasedCount** properties.

4.1.2.4.2.2 Count

Returns the amount of objects allocated in the pool.

**Visual Basic Syntax**

```
[Integer  =]  TNBXPOOL.Count
```

See also **AcquiredCount** and **ReleasedCount** properties.

4.1.2.4.2.3 Id

Specifies the pool group identification.

**Visual Basic Syntax**

```
TNBXPOOL.Id  [=  String]
```

**Remarks**

This property must be alphanumeric.

See also **Type** property, **Acquire** and **Release** methods.
4.1.2.4.2.4 ReleasedCount

Returns the amount of allocated objects that are available.

**Visual Basic Syntax**

```vbnet
[Integer =] TNBXPOOL.Count
```

See also `Count` and `AcquiredCount` properties.

4.1.2.4.2.5 Size

Specifies the maximum amount of objects for the pool.

**Visual Basic Syntax**

```vbnet
TNBXPOOL.Size [= Integer]
```

**Remarks**

When the pool size is exhausted, no more objects are allocated and calls to `Acquire` return null objects.

See also `Acquire` and `Release` methods.

4.1.2.4.2.6 Timeout

Specifies the inactivity timeout for the named pool.

**Visual Basic Syntax**

```vbnet
TNBXPOOL.Timeout [= Integer]
```

**Remarks**

Specifies the inactivity timeout for the pool. Once elapsed the specified time the timed-out telnet object is disconnected and destroyed.

See also `Type` property, `Acquire` and `Release` methods.

4.1.2.4.2.7 Type

Specifies the type of telnet objects managed by this pool.

**Visual Basic Syntax**

```vbnet
TNBXPOOL.Type [= TnbXComType]
```

**Remarks**

Available values are ct3270 and ct5250.

See also `Id` property, `Acquire` and `Release` methods.
4.1.2.4.3 Methods

4.1.2.4.3.1 Acquire

Gets a free Tnb3270X/Tnb5250X object from a pool of objects.

**VB Syntax**

```vbnet
TNBXPOOL.Acquire([sessionId as String],[InactivityTimeout as Integer])
```

**Remarks**

Gets a Tnb3270X/Tnb5250X object from the named pool (see **Id**). If there's no a free object in the named pool, it creates a new one, adds it to the pool and returns the newly created object.

If the sessionId parameter is specified, the specific associated object is returned.

InactivityTimeout parameter indicates, in minutes, how much time the telnet object will remain in the pool without activity. Default timeout is 5 minutes.

To return an object to the pool you must call Release method.

See also **Id** and **Type** property, and **Release** method.

4.1.2.4.3.2 Release

Returns a Tnb3270X/Tnb5250X object to the pool of telnet objects.

**VB Syntax**

```vbnet
TNBXPOOL.Release(tnbxxxx)
```

**Remarks**

Returns the Tnb3270X/Tnb5250X object passed as parameter to the pool of telnet objects. Used in combination with **Acquire** method.

See also **Id** and **Type** property, **Acquire** method.

4.1.2.4.3.3 ReleaseAll

Returns all the acquired objects to the pool.

**VB Syntax**

```vbnet
TNBXPOOL.ReleaseAll
```

**Remarks**

See also **Acquire** and **Release** method.
4.1.2.5 TNBXField Class Reference

4.1.2.5.1 TNBXField Class

Implements a Screen field object.

**Properties**

- Attr
- AutoEnter
- Blinking
- ByPass
- Col
- Color
- Data
- Eab
- Editable
- High
- Len
- Mandatory
- Modified
- MonoCase
- Name
- Numeric
- Pos
- Reverse
- Row
- Skip
- Underline
- Visible
- HllApiAttr
- HllApiEab

4.1.2.5.2 Properties

4.1.2.5.2.1 Attr

Sets/gets the attribute of a TNBXField.

**Visual Basic Syntax**

```vbnet
TNBXField.Attr [= Integer]
```

**Delphi Syntax**

```delphi
TNBXField.Attr [:= Integer];
```

**Remarks**

This property is for reserved use.
4.1.2.5.2.2 AutoEnter

Returns true, if an Enter AID Key should be sent when the focus of the field gets lost.

**Visual Basic Syntax**

[Boolean =] TNBXField.AutoEnter

**Delphi Syntax**

[Boolean :=] TNBXField.AutoEnter;

**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

4.1.2.5.2.3 Blinking

Returns true if the field should be shown blinking.

**Visual Basic Syntax**

[Boolean =] TNBXField.Blinking

**Delphi Syntax**


**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also [High](#), [Reverse](#) and [Underline](#) properties.

4.1.2.5.2.4 ByPass

Returns true for unprotected fields that shouldn't gain the keyboard focus.

**Visual Basic Syntax**

[Boolean =] TNBXField.ByPass

**Delphi Syntax**


**Remarks**
This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program. This property is valid only for unprotected fields.

4.1.2.5.2.5 Col
Sets/gets the screen column coordinate where the field begins.

**Visual Basic Syntax**

\[TNBXField.Col\ [\ = \ Integer]\]

**Delphi Syntax**

\[TNBXField.Col \ :\ :\ = \ Integer;\]

**Remarks**

Valid values are from 1 to \(TNBnnnnX.Cols\) property.

See Also \(Row,\ Pos\) and \(Len\) properties.

4.1.2.5.2.6 Color
Returns an integer that represents the default color corresponding to the standard or extended field attribute.

**Visual Basic Syntax**

\[Integer\ =\] \(TNBXField.Color\]

**Delphi Syntax**

\[Integer\ :\ =\] \(TNBXField.Color;\]

**Remarks**

This property returns an integer value and it’s a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also \(Blinking, High, Reverse\) and \(Underline\) properties.

4.1.2.5.2.7 Data
Sets/gets and gets the data of the field.

**Visual Basic Syntax**

\(TNBXField.Data\ [\ = \ String]\)

**Delphi Syntax**
TNBXField.Data [: = String];

Remarks
Changing the data of an unprotected field sets the Modified property to true.
See Also Modified property.

4.1.2.5.2.8 Eab
SetsGets the extended attribute of a TNBXField.

Visual Basic Syntax
[Byte =] TNBXField.Eab

Delphi Syntax
[Byte :=] TNBXField.Eab;

Remarks
This property is for reserved use.

4.1.2.5.2.9 Editable
Returns true if the field is editable (unprotected), otherwise returns false.

Visual Basic Syntax
[Boolean =] TNBXField.Editable

Delphi Syntax
[Boolean :=] TNBXField.Editable;

See Also Mandatory property.

4.1.2.5.2.10 High
Returns true if the field should be shown with a high intensity color or bold, otherwise returns false.

Visual Basic Syntax
[Boolean =] TNBXField.High

Delphi Syntax
[Boolean :=] TNBXField.High;
Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, Reverse and Underline properties.

4.1.2.5.2.11 Len

Sets/gets the field's length.

**Visual Basic Syntax**

`TNBXField.Len [= Integer]`

**Delphi Syntax**

`TNBXField.Len [:= Integer];`

See Also Row, Col and Pos properties.

4.1.2.5.2.12 Mandatory

Returns true if this field is editable and must be filled in.

**Visual Basic Syntax**

`[Boolean =] TNBXField.Mandatory`

**Delphi Syntax**

`[Boolean :=] TNBXField.Mandatory;`

**Remarks**

This property is valid only for unprotected fields.

See Also Editable property.

4.1.2.5.2.13 Modified

This property indicates if the field has been modified.

**Visual Basic Syntax**

`TNBXField.Modified [= Boolean]`

**Delphi Syntax**

`TNBXField.Modified [:= Boolean];`
See Also Data property.

### 4.1.2.5.2.14 MonoCase

This property indicates if the field accepts uppercase characters only.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBXField.MonoCase
```

**Delphi Syntax**

```delphi
[Boolean :=] TNBXField.MonoCase;
```

**Remarks**

This property is valid only for unprotected fields.

See Also Numeric property.

### 4.1.2.5.2.15 Name

Returns a string with the field name. This name is made by the telnet controls and it can be modified before first use.

**Visual Basic Syntax**

```vbnet
TNBXField.Name [= String]
```

**Delphi Syntax**

```delphi
TNBXField.Name [:= String];
```

**Remarks**

The string returned is the result of a concatenation between "R", the value of Row property, "C" and the value of Col property. For example, the name of a field located at row 4 column 30 will be R4C30.

### 4.1.2.5.2.16 Numeric

Indicates if the field accepts numeric digits only.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBXField.Numeric
```

**Delphi Syntax**

```delphi
```
Remarks

This property is valid only for unprotected fields.

See Also Monocase property.

4.1.2.5.2.17 Pos

Returns the field position in the screen.

Visual Basic Syntax

`TNBXField Pos [= Integer]`

Delphi Syntax

`TNBXField Pos [= Integer];`

Remarks

This property returns an integer ranging from 1 to $TNBnnnnX.Cols$ property multiplied by $TNBnnnnX.Rows$ property. The order of the numeration is from the left to the right and from top to bottom.

See Also Col, Row and Len properties.

4.1.2.5.2.18 Reverse

Returns true if the field should be shown in reverse video and false if the field should be shown in normal video.

Visual Basic Syntax

`[Boolean =] TNBXField.Reverse`

Delphi Syntax

`[Boolean :=] TNBXField.Reverse;`

Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, High and Underline property.

4.1.2.5.2.19 Row

Sets/gets the screen row coordinate where the field begins.

Visual Basic Syntax
TNBXField.Row [= Integer]

**Delphi Syntax**

TNBXField.Row [:= Integer];

**Remarks**

Valid values are from 1 to TNBnnnnX.Rows property.

See Also Col, Len and Pos properties.

4.1.2.5.2.20 Skip

Returns true for fields that shouldn't gain the keyboard focus.

**Visual Basic Syntax**

[Boolean ] TNBXField.Skip

**Delphi Syntax**

[Boolean ] TNBXField.Skip;

**Remarks**

When the property returns True, the cursor will skip this field preventing it to be written.

Normally, you should consider this property if you are going to implement a terminal emulation program.

4.1.2.5.2.21 Underline

Returns true if the field should be shown underlined and false if the field should not be shown underlined.

**Visual Basic Syntax**

[Boolean ] TNBXField.Underline

**Delphi Syntax**

[Boolean ] TNBXField.Underline;
Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, High and Reverse properties.

4.1.2.5.2.22 Visible

Returns true if the field is visible, otherwise it returns false.

**Visual Basic Syntax**

```
[Boolean =] TNBXField.Visible
```

**Delphi Syntax**

```
```

**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program. If the field is a label (protected) and visible is false, it shouldn't be shown. If the field is editable and visible is false, it means that this is a password field.

4.1.2.5.2.23 HllApiAttr

Returns the attribute of TNBXField in HLLAPI-compliance format.

**Visual Basic Syntax**

```
[Byte =] TNBXField.HllApiAttr
```

**Delphi Syntax**

```
[Byte :=] TNBXField.HllApiAttr;
```

**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Editable, Blinking, Color, High and Reverse properties.

4.1.2.5.2.24 HllApiEab

Returns the extended attribute of TNBXField in HLLAPI-compliance format.

**Visual Basic Syntax**
[Integer =] TNBXField.HllApiEab

**Delphi Syntax**

[Integer :=] TNBXField.HllApiEab;

**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, High and Reverse properties.

4.1.2.6 TNBXFields Class Reference

4.1.2.6.1 TNBXFields Class

Implements the Screen Fields list.

**Properties**

- **Count**
- **CursorField**
- **CursorPos**
- **Items**

**Methods**

- **Lock**

4.1.2.6.2 Properties

4.1.2.6.2.1 Count

Returns the total number of **TNBXFields** in the collection.

**Visual Basic Syntax**

[Integer =] TNBXFields.Count

**Delphi Syntax**

[Integer :=] TNBXFields.Count;

See also **Items** property.

4.1.2.6.2.2 CursorField

Returns the field where the cursor is located.

**Visual Basic Syntax**
[TNBXField =] TNBXFields.CursorField

**Delphi Syntax**

[ TNBXField := ] TNBXFields.CursorField;

See also **CursorPos** property.

### 4.1.2.6.2.3 CursorPos

Gets/sets the host-screen cursor position.

**Visual Basic Syntax**

TNBXFields.CursorPos [= Integer]

**Delphi Syntax**

TNBXFields.CursorPos [:= Integer];

**Remarks**

Valid cursor position goes from 1 to the total number of rows multiplied by the total number of columns available for the terminal type specified. The current cursor position is available in both HostFields and EditFields properties from TNB5250X/TNB3270X class. To set a new cursor position you must do it on EditFields property.

See also HostFields and EditFields property from TNB5250X/TNB3270X class.

### 4.1.2.6.2.4 Items

Contains a collection of TNBXField objects.

**Visual Basic Syntax**

[ TNBXField = ] TNB3270Fields.Items(Index As Variant)

**Delphi Syntax**

[ TNBXField := ] TNB3270Fields.Items[Index As Variant];

**Remarks**

Use Items to get an specific field from the array. The Index parameter indicates the object's index in the collection, where 0 is the index of the first element and (count - 1) is the index of the last element.

Also, you can access an specific TNBXField by its field name.

Use Items with the Count property to iterate through all the objects in the list.
See also Count and Name properties from TNBField class.

4.1.2.6.3 Methods

4.1.2.6.3.1 Lock

Used to prevent the filling of EditFields after an OnSendAid event.

**Visual Basic Syntax**

```vbnet
TNBXFields.Lock
```

**Delphi Syntax**

```delphi
TNBXFields.Lock;
```

**Remarks**

It is valid only for EditFields property from TNB5250X/TNB3270X class.

See also OnSendAid event from TNB5250X/TNB3270X class.

4.1.2.7 TnbXLPD Reference

4.1.2.7.1 TnbXLPD Control

**Properties**

- Active
- CodePage
- ConversionTable
- CurrentQueue
- Debug
- DebugDir
- DeleteFileAfterPrint
- Ebcdic
- Enabled
- Name
- Port
- PrintBanner
- PrintDocument
- QueueEnabled
- QueueName
- QueuePrinter
- Queues
- SpoolDir
- Tag
- UseFormFeedChar

**Methods**

- AboutBox
- CreateQueue
• **DeleteQueue**
• **GetControlHandle**
• **LoadFromFile**
• **SaveToFile**

**Events**

• **OnConnect**
• **OnDisconnect**
• **OnEndControlFile**
• **OnEndDataFile**
• **OnPrintBanner**
• **OnPrintDocument**
• **OnReceiveControlFile**
• **OnReceiveDataFile**
• **OnStartControlFile**
• **OnStartDataFile**

### 4.1.2.7.1.1 Properties

Controls the communication status of the TCP/IP connection. Setting it to False disables the LPD control to accept more connections nor doing any work over the communication channel.

**Visual Basic Syntax**

```
TnbXLPD.Active [= Boolean]
```

**Delphi Syntax**

```
TnbXLPD.Active [= Boolean];
```

See also **Enabled** property.

Holds the code page that will be used to translate ASCII/EBCDIC. It's initialized with "CP037/2" which is the code page for U.S.A.

**Visual Basic Syntax**

```
TnbXLPD.CodePage [= String]
```

**Delphi Syntax**
TnbXLPD.CodePage [:= String];

Remarks

CodePage and ConversionTable properties are mutually exclusive. The default setting is CodePage "CP037/2".

Examples

TnbXLPD.CodePage = "United Kingdom [CP1146]"

or

TnbXLPD.CodePage = "1146"

or

TnbXLPD.CodePage = "CP1146"

Holds complete path (Path + filename + extension) to a user defined file with the conversion table between EBCDIC and ASCII. It's initialized with an empty string ("""). If the CodePage property is used, Conversion table's value returns to an empty string. File name and extension aren't defined.

Visual Basic Syntax

TnbXLPD.ConversionTable [:= String]

Delphi Syntax

TnbXLPD.ConversionTable; [:= String]

Remarks

CodePage and ConversionTable properties are mutually exclusive. The default setting is CodePage "CP037/2".

Example

TnbXLPD.ConversionTable = "C:\MyFolder\MyFile.ebd"
Is the reference to the current queue in the LPD server. A developer could use this property to access the queue that raised the event. The property is evaluated before the event is raised, after that, its value is Nil.

**Visual Basic Syntax**

```
[TnbXQueue =] TnbXLPD.CurrentQueue
```

**Delphi Syntax**

```
[TnbXQueue :=] TnbXLPD.CurrentQueue;
```

Works in conjunction with the `DebugDir` property. When this is set to True, 2 files are generated:

- LPD_yyyymmddhhmss.SYS: Contains trace and events messages
- LPD_yyyymmddhhmss.DMP: Contains Dump data.

Usually, both files are used for technical support only.

**Visual Basic Syntax**

```
TnbXLPD.Debug [= Boolean]
```

**Delphi Syntax**

```
TnbXLPD.Debug; [= Boolean]
```

Holds the 'Directory path' where the information generated by the component is stored.

**Visual Basic Syntax**

```
TnbXLPD_DEBUGDIR [= String]
```

**Delphi Syntax**

```
TnbXLPD_DEBUGDIR; [= String]
```
Controls the possibility of a data file to be deleted after printing.

It has three states:

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldYes</td>
<td>The file is always deleted.</td>
</tr>
<tr>
<td>ldNo</td>
<td>The file is never deleted</td>
</tr>
<tr>
<td>ldClient</td>
<td>The file is deleted if the client demand to be deleted by a telnet command.</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

`TnbXLPD.DeleteFileAfterPrint [= TTnbXLPDDelete]`

**Delphi Syntax**

`TnbXLPD.DeleteFileAfterPrint; [:= TTnbXLPDDelete]`

A boolean property that controls the translate between ASCII/EBCDIC. The default value is True, so the translation will occur.

**Visual Basic Syntax**

`TnbXLPD.Ebcdic [= Boolean]`

**Delphi Syntax**

`TnbXLPD.Ebcdic; [:= Boolean]`

Controls the capability of the LPD control to process Jobs from clients.

**Visual Basic Syntax**

`TnbXLPD.Enabled [= Boolean]`

**Delphi Syntax**

`TnbXLPD.Enabled; [:= Boolean]`

See also [Active](#) property.
Returns the name used in code to identify an object.

**Visual Basic Syntax**

[String =] Object.Name

**Delphi Syntax**

[String :=] Object.Name;

Holds the port where the control accepts incoming connections from the clients. By default this port is set to the 515 port.

**Visual Basic Syntax**

TnbXLPD.Port [= Integer]

**Delphi Syntax**

TnbXLPD.Port; [= Integer]

Controls if the LPD control prints a Banner Page as the first part of each printing Job. If it's set to False, then the OnPrintBanner event isn't raised.

**Visual Basic Syntax**

TnbXLPD.PrintBanner [= Boolean]

**Delphi Syntax**

TnbXLPD.PrintBanner; [= Boolean]

Controls if the LPD control prints the document. If it's set to False, then the OnPrintDocument event isn't raised.
**Visual Basic Syntax**

`TnbXLPD.PrintDocument [ = Boolean]`

**Delphi Syntax**

`TnbXLPD.PrintDocument; [ := Boolean]`

Set and read the enabled property of the first queue in the collection. If there are no queues in the collection the LPD Component creates one.

**Visual Basic Syntax**

`TnbXLPD.QueueEnabled [ = Boolean]`

**Delphi Syntax**

`TnbXLPD.QueueEnabled; [ := Boolean]`

Set and read the name property of the first queue in the collection. If there are no queues in the collection the LPD Component creates one.

**Visual Basic Syntax**

`TnbXLPD.QueueName [ = String]`

**Delphi Syntax**

`TnbXLPD.QueueName; [ := String]`

Set and read the TnbPrinter property of the first queue in the collection. If there are no queues in the collection the LPD Component creates one.

**Visual Basic Syntax**

`TnbXLPD.QueuePrinter [ = String]`
**Delphi Syntax**

```
TnbXLPD.QueuePrinter; [:= String]
```

Holds a collection of the available Queues.

**Visual Basic Syntax**

```
[TnbXQueues =] TnbXLPD.Queues
```

**Delphi Syntax**

```
[TnbXQueues :=] TnbXLPD.Queues;
```

**Example**

```delphi
Private Sub InitQueues(MyLPD as TnbXLPD)
    If MyLPD.Queues.Count = 0 Then
        MyLPD.CreateQueue('QUEUE1')
        MyLPD.CreateQueue('QUEUE2')
    End If
End Sub
```

Holds the spool directory path where the print files are stored previously to be sent to the printer. If the property [DeleteFileAfterPrint](#) is set to False, then the file are maintained until are deleted by the user. The file extension is `.LPD`

**Visual Basic Syntax**

```
TnbXLPD.SpoolDir [= String]
```

**Delphi Syntax**

```
TnbXLPD.SpoolDir; [:= String]
```
Stores any extra data needed for your program.

**Visual Basic Syntax**

```
ObjectName.Tag [= Integer]
```

**Delphi Syntax**

```
ObjectName.Tag [= Integer];
```

When is set to True the Form Feed Character (Hex: 0C - Dec: 12) is written in the data file. When is set to False the Form Feed Character is not written.

**Visual Basic Syntax**

```
TnbXLPD.UseFormFeedChar [= Boolean]
```

**Delphi Syntax**

```
TnbXLPD.UseFormFeedChar; [= Boolean]
```

### 4.1.2.7.1.2 Methods

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```
TnbXLPD.AboutBox
```

**Delphi Syntax**

```
TnbXLPD.AboutBox;
```

Create a new Queue within the LPD control Queue list.

**Visual Basic Syntax**

```
```
**TnbXLPD.CreateQueue** (QueueName As String)

**Delphi Syntax**

*TnbXLPD.CreateQueue(QueueName:string);*

**Remarks**

Any Queue in the LPD has a unique name. When trying to create a Queue with a name that is already in use a exception occurs.

After this operation, any request from clients to use this Queue would be accepted unless the property **Enabled** is set to False.

See also **Enabled** property, **OnConnect** event.

---

Deletes a Queue within the LPD control Queue list. If the Queue has printing Jobs it won't be deleted unless ForceRemove is set to True.

**Visual Basic Syntax**

*TnbXLPD.DeleteQueue (QueueName As String, ForceRemove As Boolean)*

**Delphi Syntax**

*TnbXLPD.DeleteQueue (QueueName:string; ForceRemove:boolean);*

---

Retrieves an internal control handle for this control.

**Visual Basic Syntax**

*Long =] TnbXLPD.GetControlHandle*

**Delphi Syntax**

*[LongInt :=] TnbXLPD.GetControlHandle;*

**Remarks**

This handle is to be used in others Integration Pack ActiveX Controls.
Loads the LPD configuration from a configuration file received as a parameter.

**Visual Basic Syntax**

\[ \text{TnbXLPD.LoadFromFile (FileName As String)} \]

**Delphi Syntax**

\[ \text{TnbXLPD.LoadFromFile (FileName:string);} \]

**Example**

\[ \text{TnbXLPD.LoadFromFile('C:\MyFolder\MyConfigFile.cfg')} \]

Saves the LPD configuration in a configuration file received as a parameter.

**Visual Basic Syntax**

\[ \text{TnbXLPD.SaveToFile (FileName As String)} \]

**Delphi Syntax**

\[ \text{TnbXLPD.SaveToFile (FileName:string);} \]

**Example**

\[ \text{TnbXLPD.SaveToFile('C:\MyFolder\MyConfigFile.cfg')} \]

4.1.2.7.1.3 Events

Occurs after a successfully connection with the client is established.

**Visual Basic Declaration**

\[ \text{OnConnect(Sender as Variant, Cancel as Boolean)} \]
**Delphi Declaration**

procedure OnConnect(Sender: TObject; var Cancel: Boolean)

**Parameters**

Sender: The LPD object that raises the event.
Cancel: Controls the connection with the client, if set to FALSE the connection is closed.

See also OnDisconnect event.

Occurs after the connection with the client ends.

**Visual Basic Declaration**

OnDisconnect(Sender as Variant)

**Delphi Declaration**

procedure OnDisconnect(Sender : TObject)

**Parameters**

Sender: The LPD object that raises the event.

See also OnConnect event.

Occurs after the last control file buffer is received.

**Visual Basic Declaration**

OnEndControlFile (Sender as Variant)

**Delphi Declaration**

procedure OnEndControlFile (Sender : TObject)

**Parameters**

Sender: The LPD object that raises the event.
See also OnStartControlFile and OnReceiveControlFile events.

Occurs after the last data file buffer is received.

**Visual Basic Declaration**

OnEndDataFile (Sender as Variant)

**Delphi Declaration**

procedure OnEndDataFile (Sender : TObject)

**Parameters**

Sender: The LPD object that raises the event.

See also OnStartDataFile and OnReceiveDataFile events.

Occurs after the last banner package is sent to the printer.

**Visual Basic Declaration**

OnPrintBanner (Sender as Variant, Cancel as Boolean)

**Delphi Declaration**

procedure OnPrintBanner (Sender : TObject; var Cancel : Boolean)

**Parameters**

Sender: The LPD object that raises the event.

Cancel: If it’s TRUE, the default print method isn't used.

See also OnPrintDocument event.
Occurs after the last document package is sent to the printer.

**Visual Basic Declaration**

`OnPrintDocument (Sender as Variant, Cancel as Boolean)`

**Delphi Declaration**

procedure `OnPrintDocument (Sender : TObject; var Cancel : Boolean)`

**Parameters**

Sender: The LPD object that raises the event.  
Cancel: If it's TRUE, the default print method isn't used.

See also [OnPrintBanner](##) event.

Occurs after a control file buffer is received.

**Visual Basic Declaration**

`OnReceiveControlFile (Sender as Variant, Buffer as String, BufferSize as Integer, IsLastPacket as Boolean)`

**Delphi Declaration**

procedure `OnReceiveControlFile (Sender : TObject; Buffer : PChar; BufferSize : Integer; IsLastPacket : Boolean)`

**Parameters**

Sender: The LPD object that raises the event.  
Buffer: Received text.  
BufferSize: Buffer Size.  
IsLastPacket: If it's the last packet the value goes TRUE.

See also [OnStartControlFile](##) and [OnEndControlFile](##) events.

Occurs after a data file buffer is received.

**Visual Basic Declaration**
**OnReceiveDataFile** (Sender as Variant, Buffer as String, BufferSize as Integer, IsLastPacket as Boolean)

**Delphi Declaration**

procedure OnReceiveDataFile (Sender : TObject; Buffer : PChar; BufferSize : Integer; IsLastPacket : Boolean)

**Parameters**

Sender: The LPD object that raises the event.  
Buffer: Received text.  
BufferSize: Buffer Size.  
IsLastPacket: If it's the last packet the value goes TRUE.

**See also** [OnStartDataFile](#) and [OnEndDataFile](#) events.

Occurs after the first control file buffer is received.

**Visual Basic Declaration**

**OnStartControlFile** (Sender as Variant, Cancel as Boolean)

**Delphi Declaration**

procedure OnStartControlFile (Sender : TObject; var Cancel : Boolean)

**Parameters**

Sender: The LPD object that raises the event.  
Cancel: Controls the connection with the client, if set to TRUE the printing is aborted.

**See also** [OnReceiveControlFile](#) and [OnEndControlFile](#) events.

Occurs after the first data file buffer is received.

**Visual Basic Declaration**

**OnStartDataFile** (Sender as Variant, Cancel as Boolean)

**OnReceiveDataFile** (Sender as Variant, Buffer as String, BufferSize as Integer, IsLastPacket as Boolean)
**Delphi Declaration**

procedure OnStartDataFile (Sender : TObject; var Cancel : Boolean)

**Parameters**

Sender: The LPD object that raises the event.
Cancel: Controls the connection with the client, if set to TRUE the printing is aborted.

See also OnReceiveDataFile and OnEndDataFile events.

4.1.2.8 TnbXLPDJob Reference

4.1.2.8.1 TnbXLPDJob Class

**Properties**

- BannerClass
- BannerName
- DataFileLength
- DataFileName
- DataFileReceived
- ControlFileLength
- ControlFileName
- ControlFileReceived
- HostName
- IsConnected
- Queue
- SourceFileName
- Status
- UserName

4.1.2.8.2 Properties

4.1.2.8.2.1 BannerClass

Contains the IP address of the Class Banner received in the control file.

**Visual Basic Syntax**

[String =] TnbXLPDJob.BannerClass

**Delphi Syntax**
[String :=] TnbXLPDJob.BannerClass;

See also **BannerName** property.

### 4.1.2.8.2.2 BannerName

Contains the name of the Banner received in the control file.

**Visual Basic Syntax**

[String =] TnbXLPDJob.BannerName

**Delphi Syntax**

[String :=] TnbXLPDJob.BannerName;

See also **BannerClass** property.

### 4.1.2.8.2.3 DataFileLength

Contains the size of the data file (bytes).

**Visual Basic Syntax**

[Integer =] TnbXLPDJob.DataFileLength

**Delphi Syntax**

[Integer :=] TnbXLPDJob.DataFileLength;

See also **DataFileReceived** and **DataFileName** properties.
4.1.2.8.2.4 DataFileName

Contains the name of the data file received from the host.

**Visual Basic Syntax**

```
[String =] TnbXLPDJob.DataFileName
```

**Delphi Syntax**

```
[String :=] TnbXLPDJob.DataFileName;
```

See also [DataFileReceived](#) and [DataFileLength](#) properties.

4.1.2.8.2.5 DataFileReceived

Contains the amount of bytes of the data file received from the host.

**Visual Basic Syntax**

```
[Integer =] TnbXLPDJob.DataFileReceived
```

**Delphi Syntax**

```
[Integer :=] TnbXLPDJob.DataFileReceived;
```

See also [DataFileLength](#) and [DataFileName](#) properties.

4.1.2.8.2.6 ControlFileLength

Contains the size of the control file (bytes).

**Visual Basic Syntax**

```
[Integer =] TnbXLPDJob.ControlFileLength
```

**Delphi Syntax**

```
[Integer :=] TnbXLPDJob.ControlFileLength;
```

See also [ControlFileReceived](#) and [ControlFileName](#) properties.
4.1.2.8.2.7  ControlFileName

Contains the name of the control file received from the host.

**Visual Basic Syntax**

```vbnet
[String =] TnbXLPDJob.ControlFileName
```

**Delphi Syntax**

```pascal
[String :=] TnbXLPDJob.ControlFileName;
```

See also ControlFileReceived and ControlFileLength properties.

4.1.2.8.2.8  ControlFileReceived

Contains the amount of bytes of the control file received from the host.

**Visual Basic Syntax**

```vbnet
[Integer =] TnbXLPDJob.ControlFileReceived
```

**Delphi Syntax**

```pascal
[Integer :=] TnbXLPDJob.ControlFileReceived;
```

See also ControlFileLength and ControlFileName properties.

4.1.2.8.2.9  HostName

Contains the name of the host that requested the Job.

**Visual Basic Syntax**

```vbnet
[String =] TnbXLPDJob.HostName
```
**Delphi Syntax**

[String :=] TnbXLPDJob.HostName;

See also **UserName** and **SourceFileName** properties.

4.1.2.8.2.10 IsConnected

Indicates if the connection with the host is alive.

**Visual Basic Syntax**

[Boolean =] TnbXLPDJob.IsConnected

**Delphi Syntax**


See also **Status** property.

4.1.2.8.2.11 PrintMode

Indicates the print mode requested by the client in the control file.

Possible constant values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmText</td>
<td>Text mode</td>
</tr>
<tr>
<td>pmPostScript</td>
<td>PostScript mode</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

[TXPrinterPrintMode =] TnbXLPDJob.PrintMode

**Delphi Syntax**

[TXPrinterPrintMode :=] TnbXLPDJob.PrintMode;

See also **Status** property.
4.1.2.8.2.12 Queue

It’s a reference to the Queue which the Job belongs to.

**Visual Basic Syntax**

\[
TnbXQueue = TnbXLPDJob.Queue
\]

**Delphi Syntax**

\[
TnbXQueue := TnbXLPDJob.Queue;
\]

4.1.2.8.2.13 QueueName

It’s the name of the queue where the job was send to. Can be different to the name property in the queue object because you can use wildcards in it’s name.

**Visual Basic Syntax**

\[
String = TTnbLPDJob.QueueName
\]

**Delphi Syntax**

\[
String := TTnbLPDJob.QueueName;
\]

4.1.2.8.2.14 SourceFileName

Contains the name of the file sent by the client.

**Visual Basic Syntax**

\[
String = TnbXLPDJob.SourceFileName
\]

**Delphi Syntax**

\[
String := TnbXLPDJob.SourceFileName;
\]

See also **UserName** and **HostName** properties.
4.1.2.8.2.15 Status

Indicates the status of the communication. Each status refers to an action of the Job Class.

Possible constant values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsWaitCommand</td>
<td>Waiting a command from the host</td>
</tr>
<tr>
<td>lsWaitData</td>
<td>Receiving the data file</td>
</tr>
<tr>
<td>lsWaitControl</td>
<td>Receiving the control file</td>
</tr>
<tr>
<td>lsWaitSubcommand</td>
<td>Waiting a subcommand from the host</td>
</tr>
<tr>
<td>lsAbortJob</td>
<td>The job was aborted</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

[TxnblLPDStatus =] TnbXLPDJob.Status

**Delphi Syntax**


See also IsConnected property.

4.1.2.8.2.16 UserName

Contains the user name that requested the Job.

**Visual Basic Syntax**

[String =] TTNBXLPDJob.UserName

**Delphi Syntax**

[String :=] TTNBXLPDJob.UserName;

See also SourceFileName and HostName properties.
4.1.2.8.3 Constants

4.1.2.8.3.1 TxPrinterPrintMode

These values are used in the PrintMode property of TnbXLDPDJob class.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>pmText</td>
<td>Text mode</td>
</tr>
<tr>
<td>pmPostScript</td>
<td>PostScript mode</td>
</tr>
</tbody>
</table>

4.1.2.8.3.2 TxTnbLPDStatus

These values are used in the Status property of TnbXLDPDJob class.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsWaitCommand</td>
<td>Waiting a command from the host</td>
</tr>
<tr>
<td>lsWaitData</td>
<td>Receiving the data file</td>
</tr>
<tr>
<td>lsWaitControl</td>
<td>Receiving the control file</td>
</tr>
<tr>
<td>lsWaitSubcommand</td>
<td>Waiting a subcommand from the host</td>
</tr>
<tr>
<td>lsAbortJob</td>
<td>The job was aborted</td>
</tr>
</tbody>
</table>

4.1.2.9 TnbXLDPDJobs Reference

4.1.2.9.1 TnbXLDPDJobs Collection

Properties

- Items

Methods

- Count
- IndexOf
4.1.2.9.2 Properties
4.1.2.9.2.1 Items

Allows the access to an specific Job.

**Visual Basic Syntax**

\[ TnbXLPDJob =] TnbXLPDJobs.Items (Index as Integer) \]

**Delphi Syntax**

\[ TnbXLPDJob :=] TnbXLPDJobs.Items (Index as Integer); \]

4.1.2.9.3 Methods
4.1.2.9.3.1 Count

Returns the amount of Jobs within the collection.

**Visual Basic Syntax**

\[ Integer =] TnbXLPDJobs.Count \]

**Delphi Syntax**

\[ Integer :=] TnbXLPDJobs.Count; \]

See also **Items** property.

4.1.2.9.3.2 IndexOf

Returns the index of a TnbXLPDJob object within the collection.

**Visual Basic Syntax**

\[ Integer =] TnbXLPDJobs.IndexOf (Item as TTnbLPDJob) \]

**Delphi Syntax**

\[ Integer :=] TnbXLPDJobs.IndexOf (Item:TTnbLPDJob); \]
4.1.2.10 TnbXLPDQueue Reference

4.1.2.10.1 TnbXLPDQueue Class

Properties

- CurrentJob
- DeleteFileAfterPrint
- Ebcdic
- Enabled
- Jobs
- LPD
- Name
- PrintBanner
- PrintDocument
- SpoolDir

Methods

- GetControlHandle
- Show_printerEditor

4.1.2.10.2 Properties

4.1.2.10.2.1 CurrentJob

Holds a pointer to the current Job. It’s assigned before calling an event and after that it’s value is Null.

Visual Basic Syntax

[TnbXLPDQueue =] TnbXLPDQueue.CurrentJob

Delphi Syntax


Example

Dim Job as TnbXLPDJob
....
Job = TnbXLPDQueue.CurrentJob
4.1.2.10.2.2  DeleteFileAfterPrint

Controls the possibility of a data file to be deleted after printing.

It has three states:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>IdYes</td>
<td>The file is always deleted</td>
</tr>
<tr>
<td>IdNo</td>
<td>The file never is deleted</td>
</tr>
<tr>
<td>IdClient</td>
<td>The file is deleted if the client demands to be deleted by a telnet command</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

```
TnbXLPDQueue.DeleteFileAfterPrint [= TxBnbXLPDDelete]
```

**Delphi Syntax**

```
TnbXLPDQueue.DeleteFileAfterPrint; [= TxBnbXLPDDelete]
```

4.1.2.10.2.3  Ebcdic

A boolean property that controls the translate between ASCII/EBCDIC. The default value is TRUE, so the translation will occur.

**Visual Basic Syntax**

```
TnbXLPDQueue.Ebcdic [= Boolean]
```

**Delphi Syntax**

```
TnbXLPDQueue.Ebcdic; [= Boolean]
```

4.1.2.10.2.4  Enabled

Controls the capability of the LPD control to process Jobs from clients.

**Visual Basic Syntax**
4.1.2.10.2.5 Jobs

Contains a collection of TnbXLPDJob objects, one for each job in the queue.

**Visual Basic Syntax**

\[
[TnbXLPDJob =] TnbXLPDQueue.Jobs
\]

**Delphi Syntax**

\[
\]

**Example**

Dim Job as TnbXLPDJob
...
for i=0 to TnbXLPDQueue.Jobs.Count
    Job = TnbXLPDQueue.Jobs[i]
    'do something with the Job object
next

4.1.2.10.2.6 LPD

Contains a pointer to the LPD object.

**Visual Basic Syntax**

\[
[TnbLPD =] TnbXLPDQueue.LPD
\]

**Delphi Syntax**

\[
[TnbLPD :=] TnbXLPDQueue.LPD;
\]
4.1.2.10.2.7 Name

Is the name that identifies the Queue. You can use wildcards to cover a range of queues.

**Wildcards:**

<table>
<thead>
<tr>
<th>Wildcard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'*'</td>
<td>Can be replaced with any string</td>
</tr>
<tr>
<td>'?'</td>
<td>Can be replaced with any character</td>
</tr>
</tbody>
</table>

**Note:** Only one '*' is allowed in each name.

**Examples:**

```
TnbXLPDQueue.Name = "COL*"
TnbXLPDQueue.Name = "C?L*"
TnbXLPDQueue.Name = "L?AS?AA"
```

**Visual Basic Syntax**

```
[String =] TnbXLPDQueue.Name
```

**Delphi Syntax**

```
[String :=] TnbXLPDQueue.Name;
```

4.1.2.10.2.8 PrintBanner

Determines if a banner page is going to be printed at the top of each Job.

**Visual Basic Syntax**

```
TnbXLPDQueue.PrintBanner [= Boolean]
```

**Delphi Syntax**

```
TnbXLPDQueue.PrintBanner; [= Boolean]
```

4.1.2.10.2.9 PrintDocument

Determines if the document is going to be printed.

**Visual Basic Syntax**

```
TTNBXLPDQueue.PrintDocument [= Boolean]
```
Delphi Syntax

\[ TTNBXLPDQueue.PrintDocument; [:= Boolean] \]

4.1.2.10.2.10 SpoolDir

Holds the Queue's spool folder path.

Visual Basic Syntax

\[ TnbXLPDQueue.SpoolDir [= String] \]

Delphi Syntax

\[ TnbXLPDQueue.SpoolDir; [:= String] \]

4.1.2.10.3 Methods

4.1.2.10.3.1 GetControlHandle

Retrieves an internal control handle for this control.

Visual Basic Syntax

[Long =] TnbXLPDQueue.GetControlHandle

Delphi Syntax


Remarks

This handle is to be used in others Integration Pack ActiveX Controls.

4.1.2.10.3.2 ShowPrinterEditor

Show an standard dialog to edit printer properties.
Visual Basic Syntax

TTnbXLPDQueue.ShowPrinterEditor

Delphi Syntax

TTnbXLPDQueue.ShowPrinterEditor;

4.1.2.10.4 Constants

4.1.2.10.4.1 TxTnbXLPDDelete

These values are used in the **DeleteFileAfterPrint** property of **TnbXLPDQueue** class.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldYes</td>
<td>The file is always deleted</td>
</tr>
<tr>
<td>ldNo</td>
<td>The file never is deleted</td>
</tr>
<tr>
<td>ldClient</td>
<td>The file is deleted if the client demands to be deleted by a telnet command</td>
</tr>
</tbody>
</table>

4.1.2.11 TnbXLPDQueues Reference

4.1.2.11.1 TnbXLPDQueues Collection

**Properties**

- **Items**
- **Count**

4.1.2.11.2 Properties

4.1.2.11.2.1 Items

Allows access to an specific queue.

**Visual Basic Syntax**

[TnbXLPDQueue =] TnbXLPDQueues.Items (Index as Integer)
**Delphi Syntax**

```
[TnbXLPDQueue :=] TnbXLPDQueues.Items(Index as Integer);
```

### 4.1.2.11.2.2 Count

Holds the amount of Queues in the collection.

**Visual Basic Syntax**

```
[Integer :=] TnbXLPDQueues.Count
```

**Delphi Syntax**

```
[Integer :=] TnbXLPDQueues.Count;
```

### 4.1.2.12 TNBXIndFile Reference

#### 4.1.2.12.1 TNBXIndFile Control

**Properties**

- AbortString
- Append
- Ascii
- BlkSize
- BufSize
- Bytes
-CrLf
- Direction
- FtCommand
- FtMode
- FtName
- HostFileName
- HostKind
- LocalFileName
- Lrecl
- PrimarySpace
- ProfilesControl
- RecFm
- SecSpace
- ShowDialog
- ShowEditor
- Stream
• SubKey
• TelnetControl
• TimeOut
• Units

Methods

• AboutBox
• Abort
• AsVariant
• LoadFromXMLFile
• Receive
• Run
• SaveToXMLFile
• Send

Events

• OnAborting
• OnComplete
• OnRunning
• OnUpdateLength

4.1.2.12.2 Properties

4.1.2.12.2.1 AbortString

This property contains a string that describes why the file transfer was aborted.

**Visual Basic Syntax**

[String =] TNBXIndFile.AbortString

**Delphi Syntax**

[String :=] TNBXIndFile.AbortString;

See also **Abort** method.

4.1.2.12.2.2 Append

If set to True, the file to be transferred is appended to the existing one. If the destination file doesn't exist (HostFileName in case of Send method or LocalFileName in case of
Receive method), it is treated as new. If set to False, no append is taken place. If the destination file exists, it is replaced.

**Visual Basic Syntax**

`TNBXIndFile.Append [= Boolean]`

**Delphi Syntax**

`TNBXIndFile.Append; [:= Boolean]`

4.1.2.12.2.3 Ascii

It is used when transferring files from host to PC to convert EBCDIC characters to Ascii characters.

**Visual Basic Syntax**

`TNBXIndFile.AscII [= Boolean]`

**Delphi Syntax**

`TNBXIndFile.AscII; [:= Boolean]`

See also **Crlf** property.

4.1.2.12.2.4 BlkSize

Is the block size to be used for the host destination file (**HostFileName** property) when a new file is created in **Send** method or **Run** method with **Direction**=0 indicator set. Must be a multiple of record length (**Lreci** property), if fixed blocked (**RecFm** property) file format is specified.

**Visual Basic Syntax**

`TNBXIndFile.BlkSize [= Integer]`

**Delphi Syntax**

`TNBXIndFile.BlkSize; [:= Integer]`
See also **HostFileName** property, **Lrecl** property, **RecFm** property, **Run** method, **Send** method.

### 4.1.2.12.2.5 BufSize

This property has the size of the buffer used to do the file transfer. It's only used when the connection mode is DFT.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.BufSize [= Integer]
```

**Delphi Syntax**

```delphi
TNBXIndFile.BufSize; [= Integer]
```

See also **Stream** property, **Bytes** property.

### 4.1.2.12.2.6 Bytes

Is the current bytes transferred to the destination file, **HostName** in **Send** or **Run** (**Direction**=0) methods, or **LocalFileName** in **Receive** or **Run** (**Direction**=1) methods.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.Bytes [Integer =]
```

**Delphi Syntax**

```delphi
TNBXIndFile.Bytes; [Integer :=]
```

See also **OnUpdateLength** event.
4.1.2.12.2.7 CrLf

It is used when transferring file from a host to a PC.
If set to True, CRLF characters, chr(13)+chr(10), are added to the end of each line. If set to False no characters are added.

**Visual Basic Syntax**

```
TNBXIndFile.CrLf [= Boolean]
```

**Delphi Syntax**

```
TNBXIndFile.CrLf; [= Boolean]
```

See also **Ascii** property.

4.1.2.12.2.8 Direction

This property is used in conjunction with the Run method to indicate the transfer direction of the File Transfer.
A 0 value indicates a send direction (from LocalFileName file name to HostFileName file name).
A 1 value indicates a receive direction (from HostFileName file name to LocalFileName file name).

**Visual Basic Syntax**

```
TNBXIndFile.Direction [= Integer]
```

**Delphi Syntax**

```
TNBXIndFile.Direction; [= Integer]
```

See also **Run** method.

4.1.2.12.2.9 FtCommand

Sets/gets the name of the file transfer command in the host system, usually "IND$FILE".
This property doesn't have to be changed from its default value.

**Visual Basic Syntax**

```
```
TNBXIndFile.FtCommand [= IND$FILE]

**Delphi Syntax**

TNBXIndFile.FtCommand; [= IND$FILE]

4.1.2.12.2.10 FtMode

Indicates the File Transfer Mode. By default the value is ftDFT, but if the host can't handle DFT mode it automatically changes to ftCut.

**Visual Basic Syntax**

[String =] TNBXIndFile.FtMode

**Delphi Syntax**

[String :=] TNBXIndFile.FtMode;

**Remarks**

Possible values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftDFT</td>
<td>DFT mode</td>
</tr>
<tr>
<td>ftCut</td>
<td>Cut mode</td>
</tr>
</tbody>
</table>

See also **TxFtMode** constants.

4.1.2.12.2.11 FtName

It's the name of the file transfer session.

**Visual Basic Syntax**

TNBXIndFile.FtName [= String]

**Delphi Syntax**

TNBXIndFile.FtName; [:= String]
4.1.2.12.2.12 HostFileName

Sets/gets the name in the host system of the file to be transferred. It is used either in the **Send** / **Receive** methods mode or the **Run** method mode. The file name syntax has to be valid for the host system.

**Example:**

"File Name a1"  (file name, file type, file mode)
valid name for VM/CMS.

"sys1.vtamlst(atcstr00)" valid name for PDS member name under Tso.

"sys2.user.dat" valid name for catalogued file.

**Visual Basic Syntax**

TNBXIndFile.HostFileName [= FileName.dat]

**Delphi Syntax**

TNBXIndFile.HostFileName; [= FileName.dat]

See also **LocalFileName** property.

4.1.2.12.2.13 HostKind

Sets/gets the environment that the host supports.

**Visual Basic Syntax**

TNBXIndFile.HostKind = TxHostKind

**Delphi Syntax**

TNBXIndFile.HostKind := TxHostKind;

**Remarks**
Possible values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hVM</td>
<td>VM</td>
</tr>
<tr>
<td>hTSO</td>
<td>TSO</td>
</tr>
<tr>
<td>hCics</td>
<td>Cics</td>
</tr>
</tbody>
</table>

See also **TxHostKind** constants.

### 4.1.2.12.2.14 LocalFileName

Sets/gets the name in the local system of the file to be transferred. It is used either in the **Send** / **Receive** methods mode or the **Run** method mode. The file name syntax has to be valid in the local system.

**Example:**

**Visual Basic Syntax**

```visualbasic
TNBXIndFile.LocalFileName = FileName.dat
```

**Delphi Syntax**

```delphi
TNBXIndFile.LocalFileName := FileName.dat
```

See also **HostFileName** property.

### 4.1.2.12.2.15 LRecl

Sets/gets the record length to be used for the host destination file (**HostFileName** property) when a new file is created during the execution of a **Send** method or **Run** method (with **Direction**=0 indicator set).

It must be a submultiple of block size (**BlkSize** property), if fixed blocked (**RecFm** property) file format is specified.

**Visual Basic Syntax**

```visualbasic
TNBXIndFile.LRecl = Integer
```

**Delphi Syntax**

```delphi
TNBXIndFile.LRecl := Integer
```
4.1.2.12.2.16 PrimSpace

Sets/gets the primary space quantity used to allocate the host HostFileName file in the Units selected, when a new file is created during the execution of a Send method or Run method (with Direction=0 indicator set).

**Visual Basic Syntax**

TNBXIndFile.PrimSpace [= Integer]

**Delphi Syntax**

TNBXIndFile.PrimSpace; [= Integer]

See also HostFileName property, BlkSize property, RecFm property, Run method, Send method.

4.1.2.12.2.17 ProfilesControl

Sets/gets the ProfilesControl Component as the profile manager for this Component.

**Visual Basic Syntax**

TNBXIndFile.ProfilesControl [= ProfilesControl]

**Delphi Syntax**

TNBXIndFile.ProfilesControl; [= ProfilesControl]

After setting this property, you can access to a collection of connections to be generated and customized through the ShowEditor method.

See also TnbXProfile reference, SubKey property, ShowEditor property.
4.1.2.12.2.18 RecFm

Sets/gets the record format of the host HostFileName file.

**Visual Basic Syntax**

TNBXIndFile.RecFm [= Integer]

**Delphi Syntax**

TNBXIndFile.RecFm; [= Integer]

Possible values for this property are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default file format</td>
</tr>
<tr>
<td>1</td>
<td>Fixed file format</td>
</tr>
<tr>
<td>2</td>
<td>Variable file format</td>
</tr>
<tr>
<td>3</td>
<td>Undefined file format</td>
</tr>
</tbody>
</table>

See also BlkSize property, HostFileName property, PrimSpace property, Run method, SecSpace property, Send method, Units property.

4.1.2.12.2.19 SecSpace

Sets/gets the secondary space quantity used to allocate the host HostFileName file in the Units selected, when a new file is created during the execution of a Send method or Run method (with Direction=0 Send indicator set).

**Visual Basic Syntax**

TNBXIndFile.SecSpace [= Integer]

**Delphi Syntax**

TNBXIndFile.SecSpace; [= Integer]

See also HostFileName property, PrimSpace property, BlkSize property, RecFm property, Units property, Run method, Send method.
4.1.2.12.2.20 ShowDialog

Specifies if the file transfer "Bytes Transferred" dialog appears in each buffer sent to the destination. This dialog could be used to Cancel the file transfer in progress.

Visual Basic Syntax

TNBXIndFile.ShowDialog [= Boolean]

Delphi Syntax

TNBXIndFile.ShowDialog; [= Boolean]

See also OnAborting event.

4.1.2.12.2.21 ShowEditor

Specifies if the File Transfer Settings dialog appears previously to each file transfer request to let the user modify the file transmission parameters. All the fields in both Transfers and Advanced tabs have the corresponding property that can be set programmatically instead than doing it through this dialog.

Visual Basic Syntax

TNBXIndFile.ShowEditor [= Boolean]

Delphi Syntax

TNBXIndFile.ShowEditor; [= Boolean]
4.1.2.12.2.22 Stream

Sets/gets a byte stream that contains the information the PC received from host or the information the PC will send to the host.

**Visual Basic Syntax**

`TNBXIndFile.Stream(Buffer As String, BufSize As Integer)`

**Delphi Syntax**

`TNBXIndFile.Stream(Buffer: string; BufSize: integer);`

See also `BufSize` property, `Bytes` property.

4.1.2.12.2.23 SubKey

Sets/gets the subkey under which the connection profiles will be stored.

**Visual Basic Syntax**

`TNBXIndFile.SubKey [= String]`

**Delphi Syntax**

`TNBXIndFile.SubKey; [:= String]`

**Remarks**

Default value is "TNB3270E" for TNBX3270E Component and "TNB5250" for TNBX5250 Component.

See also `ProfilesControl` property, `FtName` property.

4.1.2.12.2.24 TelnetControl

Sets/gets the TTNB3270E or TTNB5250 Component as the telnet client Component.
4.1.2.12.2.25 TimeOut

Sets/gets the time slice to wait for the host connection to be established.

**Visual Basic Syntax**

\[ TNBXIndFile.TelnetControl \[= TNBnnnnX] \]

**Delphi Syntax**

\[ TNBXIndFile.TelnetControl; [:= TNBnnnnX] \]

See also **Receive** method, **Run** method, **Send** method.

4.1.2.12.2.26 Units

Sets/gets the type of allocation unit of the host **HostFileName** file used when a new file is created during the execution of a **Send** method or **Run** method (with **Direction**=0 Send indicator set).

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default unit of allocation</td>
</tr>
<tr>
<td>1</td>
<td>Tracks unit of allocation</td>
</tr>
<tr>
<td>2</td>
<td>Cylinder unit of allocation</td>
</tr>
<tr>
<td>3</td>
<td>Average Blocks unit of allocation</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

\[ TNBXIndFile.Units \[= Integer] \]

**Delphi Syntax**

\[ TNBXIndFile.Units; [:= Integer] \]
See also **BlkSize** property, **PrimarySpace** property, **RecFm** property, **SecondarySpace** property.

4.1.2.12.3 Methods

4.1.2.12.3.1 AboutBox

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.AboutBox
```

**Delphi Syntax**

```delphi
TNBXIndFile.AboutBox;
```

4.1.2.12.3.2 Abort

The transfer is interrupted after changing the current transfer's status to abort.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.Abort()
```

**Delphi Syntax**

```delphi
TNBXIndFile.Abort();
```

See also **AbortString** property, **OnAborting** event.

4.1.2.12.3.3 AsVariant

Returns the object as variant variable type.

**Visual Basic Syntax**

```vbnet
```
4.1.2.12.3.4 LoadFromXMLFile

Loads the XML code corresponding to the `TNBXHotSpots` object.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.LoadFromXMLFile (XMLFile As String)
```

**Delphi Syntax**

```delphi
TNBXIndFile.LoadFromXMLFile (XMLFile: string);
```

See also `SaveToXMLFile` method.

4.1.2.12.3.5 Receive

This method is used to receive into `LocalFileName` file the `HostFileName` file.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.Receive()
```

**Delphi Syntax**

```delphi
TNBXIndFile.Receive();
```

See also `Direction` property, `Run` method, `Send` method.
4.1.2.12.3.6 Run

Is used to send `LocalFileName` file from the PC to the `HostFileName` in the host if `Direction` property is set to 0=Send, or vice versa if `Direction` property is set to 1=Receive.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.Run()
```

**Delphi Syntax**

```delphi
TNBXIndFile.Run();
```

See also `Direction` property, `Receive` method, `Send` method.

4.1.2.12.3.7 SaveToXMLFile

Exports the XML code corresponding to the `TNBXHotSpots` object.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.SaveToXMLFile (XMLFile As String)
```

**Delphi Syntax**

```delphi
TNBXIndFile.SaveToXMLFile (XMLFile: string);
```

See also `LoadFromXMLFile` method.

4.1.2.12.3.8 Send

This method is used to send `LocalFileName` file from the PC to the `HostFileName` in the host using the properties directly modified or using the Editor dialog to fill them interactively.

**Visual Basic Syntax**

```vbnet
TNBXIndFile.Send()
```

**Delphi Syntax**

```delphi
TNBXIndFile.Send();
```
See also Direction property, Run method, Receive method.

4.1.2.12.4 Events

4.1.2.12.4.1 OnAborting

Occurs after the Cancel button in the 'Bytes Transferred' dialog is pressed indicating a user defined cancel operation.

Declaration

procedure OnAborting (Sender: TObject)

Parameters

Sender: The TNBXIndFile object that raises the event.

See also OnComplete event.

4.1.2.12.4.2 OnComplete

Occurs after the file transfer operation is finished.

Declaration

procedure OnComplete (Sender: TObject)

Parameters

Sender: The TNBXIndFile object that raises the event.

See also OnAborting event.

4.1.2.12.4.3 OnRunning

Occurs when the file transfer operation has been started.

Declaration
procedure OnRunning (Sender: TObject)

**Parameters**

Sender: The *TNBXIndFile* object that raises the event.

*See also* [OnComplete](#) event.

### 4.1.2.12.4.4 OnUpdateLength

Occurs every time a buffer of data of *BufSize* bytes are sent from the source file to the destination file. Previously this event is fired the *Bytes* property is updated accordingly to reflect the total amount of data sent.

**Declaration**

procedure OnUpdateLength (Sender: TObject)

**Parameters**

Sender: The *TNBXIndFile* object that raises the event.

*See also* [OnRunning](#) event.

### 4.1.2.12.5 Constants

#### 4.1.2.12.5.1 TfMode

These values are used in the *FTMode* property of *TnbXIndFile* control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftDFT</td>
<td>DFT mode</td>
</tr>
<tr>
<td>ftCut</td>
<td>Cut mode</td>
</tr>
</tbody>
</table>

#### 4.1.2.12.5.2 TxHostKind

These values are used in the *HostKind* property of *TNBXIndFile* control.
### 4.1.2.13 Constants

#### 4.1.2.13.1 TNBSSLMethod

These values are used for the **SSLMethod** property of TNB5250X control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>metSSL23</td>
<td>Supports SSL 2.0 and 3.0 version.</td>
</tr>
<tr>
<td>metSSL2</td>
<td>Supports SSL 2.0 version.</td>
</tr>
<tr>
<td>metSSL3</td>
<td>Supports SSL 3.0 version.</td>
</tr>
<tr>
<td>metTLS1</td>
<td>Supports TLS 1.0 version.</td>
</tr>
</tbody>
</table>

#### 4.1.2.13.2 TNBXCP

These values are used in the **CodePage** property of telnet controls.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada</td>
<td>Canada</td>
</tr>
<tr>
<td>cpCzechRepublic</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cpDenmark</td>
<td>Denmark</td>
</tr>
<tr>
<td>cpFinland</td>
<td>Finland</td>
</tr>
<tr>
<td>cpFrance</td>
<td>France</td>
</tr>
<tr>
<td>cpGermany</td>
<td>Germany</td>
</tr>
<tr>
<td>cpGreece</td>
<td>Greece</td>
</tr>
<tr>
<td>cpGreece2</td>
<td>Greece</td>
</tr>
<tr>
<td>cpItaly</td>
<td>Italy</td>
</tr>
<tr>
<td>cpLatinAmerica</td>
<td>Latin America</td>
</tr>
<tr>
<td>cpNetherlands</td>
<td>Netherlands</td>
</tr>
<tr>
<td>cpNorway</td>
<td>Norway</td>
</tr>
<tr>
<td>cpPoland</td>
<td>Poland</td>
</tr>
<tr>
<td>cpPortugal</td>
<td>Portugal</td>
</tr>
<tr>
<td>cpSpain</td>
<td>Spain</td>
</tr>
<tr>
<td>spSweden</td>
<td>Sweden</td>
</tr>
<tr>
<td>cpSwitzerlandFrench</td>
<td>Switzerland (French)</td>
</tr>
<tr>
<td>cpSwitzerlandGerman</td>
<td>Switzerland (German)</td>
</tr>
<tr>
<td>cpUnitedKingdom</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>Constant Value</td>
<td>Meaning</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>cpUnitedStates</td>
<td>United States</td>
</tr>
<tr>
<td>cpInternational</td>
<td>International</td>
</tr>
</tbody>
</table>

### 4.1.2.13.3 TNBXAid

These values are used in the **AidKey** property of telnet controls.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>akPF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>akPF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>akPF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>akPF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>akPF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>akPF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>akPF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>akPF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>akPF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>akPF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>akPF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>akPF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>akPF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>akPF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>akPF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>akPF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>akPF24</td>
<td>PF24 key</td>
</tr>
</tbody>
</table>

### 4.1.2.13.4 TNBXSS

These values are used in the **SessionState** property of telnet controls.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssNoSession</td>
<td>Not in session</td>
</tr>
</tbody>
</table>
4.1.2.13.5 TNBX5250TT

These values are used in the `TerminalType` property of TNBX5250 control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt5211m2 = 0</td>
<td>IBM-5211-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3179m2 = 1</td>
<td>IBM-3179-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3477mFC = 2</td>
<td>IBM-3477-FC 27 rows x 132 columns</td>
</tr>
<tr>
<td>tt3180m2 = 3</td>
<td>IBM-3180-2 27 rows x 132 columns</td>
</tr>
</tbody>
</table>

4.1.2.13.6 TNBX3270TT

These values are used in the `TerminalType` property of TNBX3270 control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt3278m2 = 0</td>
<td>IBM-3278-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m2E = 1</td>
<td>IBM-3278-2-E 24 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m3 = 2</td>
<td>IBM-3278-3 32 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m3E = 3</td>
<td>IBM-3278-3-E 32 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m4 = 4</td>
<td>IBM-3278-4 43 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m4E = 5</td>
<td>IBM-3278-4-E 43 rows x 80 columns extended</td>
</tr>
</tbody>
</table>

4.1.3 Terminal Emulator Controls

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack ActiveX Terminal Emulator Controls.

4.1.3.1 TNBXEmulator Control

4.1.3.1.1 TNBXEmulator Control

This is an implementation of the screen-emulation panel as an ActiveX Control without the keyboard interface.

**Properties**

- `AidHookControl`
- `CursorPos`
- `DisableKbdLockOnError`
- `EnableSelection`
- `ErrorLine`
- `FontAutoSize`
- `HotSpotsControl`
- `InputInhibitEnabled`
• InsertMode
• IsInputInhibited
• LeftMargin
• ProfilesControl
• RulerType
• StyleName
• SubKey
• TelnetControl
• TopMargin
• Visible

Methods

• AboutBox
• AidKeyPressed
• AsVariant
• ClassNameIs
• Copy
• GetScreenRowEx
• GetScreenText
• LoadFromFile
• LocalKeyPressed
• Paste
• PrintScreen
• PutFields
• SaveToFile
• SendKeys
• ShowEditor

Events

• OnClick
• OnCursorPos
• OnDbClick
• OnInhibitChange
• OnInsertModeChange

Constants

• TxRulerType

4.1.3.1.2 Properties

4.1.3.1.2.1 AidHookControl

Sets the TNBXAidHook ActiveX Control as the keyboard handler for this control.

Visual Basic Syntax

TNBXEmulator.AIDHookControl [ = TNBXAidHook]

Delphi Syntax
TNBXEmulator.AIDHookControl [:= TNBXAidHook];

Remarks

Normally, to send information to the host through the TNBXEmulator you must call the PutFields method and then send the properly AID key to the telnet control (see AidKey property or SendAid method of TNB3270X/TNB5250X Controls). Assigning this property to a TNBXAidHook Control, you don't need to do these calls. When the TNBXEmulator receives a function key from TNBXAidHook, process it internally, sending the modified fields and the AID key pressed to the host.

See Also TNBXAidHook control.

4.1.3.1.2.2 Ctl3d

Sets/gets the 3D control's appearance.

Visual Basic Syntax

TNBXEmulator.Ctl3d [= Boolean]

Delphi Syntax

TNBXEmulator.Ctl3d [:= Boolean];

Remarks

Default value is False

4.1.3.1.2.3 CursorPos

Gets/sets the host-screen cursor position.

Visual Basic Syntax

TNBXEmulator.CursorPos [= Integer]

Delphi Syntax

TNBXEmulator.CursorPos [:= Integer];

Remarks

Valid cursor position goes from 1 to the total number of rows multiplied by the total number of columns available for the terminal type specified. The current cursor position is available in both HostFields and EditFields properties from TNB5250X/TNB3270X class. To set a new cursor position you must do it on EditFields property.

See also HostFields and EditFields property from TNB5250X/TNB3270X class.
4.1.3.1.2.4 DisableKbdLockOnError

Set/gets a boolean value to lock/unlock the keyboard when the host returns an error message.

**Visual Basic Syntax**

```
TNBXEmulator.DisableKbdLockOnError [= Boolean]
```

**Delphi Syntax**

```
TNBXEmulator.DisableKbdLockOnError [:= Boolean];
```

**Remarks**

Default value is **False**.

See Also **ErrorLine** property.

4.1.3.1.2.5 EnableSelection

Enables/disables the selection of an emulation-screen area for copying it to the clipboard.

**Visual Basic Syntax**

```
TNBXEmulator.EnableSelection [= Boolean]
```

**Delphi Syntax**

```
TNBXEmulator.EnableSelection [:= Boolean];
```

**Remarks**

Set the property to true if you plan to use **Copy** method, to allow the selection of a copy area.

Default value is **True**.

See Also **Copy** and **Paste** methods.

4.1.3.1.2.6 ErrorLine

Set/gets the line number where the error message will be displayed.

**Visual Basic Syntax**

```
TNBXEmulator.ErrorLine [= Integer]
```

**Delphi Syntax**
4.1.3.1.2.7  FontAutoSize

Set/goes the autosize state of the emulator-screen font.

**Visual Basic Syntax**

TNBXEmulator.FontAutoSize [ = Boolean]

**Delphi Syntax**

TNBXEmulator.FontAutoSize [ := Boolean];

**Remarks**

Default value is True.

4.1.3.1.2.8  HotSpotsControl

Sets the TNBXHotSpot Control as the hotspots handler for this control.

**Visual Basic Syntax**

TNBXEmulator.HotSpotsControl [ = TNBXHotSpot]

**Delphi Syntax**

TNBXEmulator.HotSpotsControl [ := TNBXHotSpot];

See Also TNBXHotSpot control.

4.1.3.1.2.9  InputInhibitEnabled

Enables/disables the input inhibit condition when any key is pressed on a protected field.

**Visual Basic Syntax**

TNBXEmulator.ErrorLine [ := Integer];
TNBXEmulator.InputInhibitEnabled [ = Boolean]

**Delphi Syntax**

```delphi
TNBXEmulator.InputInhibitEnabled [: = Boolean];
```

**Remarks**

Default value is `True`.

See Also `IsInputInhibited` property and `OnInputInhibitChange` event.

### 4.1.3.1.2.10 InsertMode

Sets/gets the insert/overwrite keyboard mode for typing on unprotected fields.

**Visual Basic Syntax**

```vbnet
TNBXEmulator.InsertMode [ = Boolean]
```

**Delphi Syntax**

```delphi
TNBXEmulator.InsertMode [: = Boolean];
```

**Remarks**

Default value is `False`.

See Also `OnInsertModeChange` event.

### 4.1.3.1.2.11 IsInputInhibited

Returns true if the input of the emulator-screen is inhibited.

**Visual Basic Syntax**

```
[Boolean =] TNBXEmulator.IsInputInhibited
```

**Delphi Syntax**

```
[Boolean :=] TNBXEmulator.IsInputInhibited;
```

See Also `InputInhibitEnabled` property and `OnInputInhibitChange` event.
4.1.3.1.2.12 LeftMargin

Sets emulation screen left margin.

**Visual Basic Syntax**

`TNBXEmulator.LeftMargin [= Integer]`

**Delphi Syntax**

`TNBXEmulator.LeftMargin := Integer;`

**Remarks**

Left margin emulation value is measured in pixels.

4.1.3.1.2.13 Profiles

Sets the `TNBXProfiles` ActiveX Control as the profile manager for this control.

**Visual Basic Syntax**

`TNBXEmulator.Profiles [= TNBXProfiles] Profiles ???`

**Delphi Syntax**

`TNBXEmulator.Profiles [= TNBXProfiles];`

**Remarks**

After setting this property, you can access a collection of styles to be generated and customized through the `ShowEditor` method.

See Also `TNBXProfiles` control, `StyleName` property and `ShowEditor` method.

4.1.3.1.2.14 RulerType

Sets/gets the format style of the ruler. This ruler shows the cursor position.

**Visual Basic Syntax**

`TNBXEmulator.RulerType [= TxRulerType]`

**Delphi Syntax**
\texttt{TNBXEmulator.RulerType} [:= \texttt{TxRulerType}];

**Remarks**

For example, \textit{ruCrosshair} constant value will draw one vertical and one horizontal lines, and the intersection will be the cursor position.

Valid constant values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruNone = 0</td>
<td>No lines</td>
</tr>
<tr>
<td>ruCrosshair = 1</td>
<td>Horizontal and vertical lines</td>
</tr>
<tr>
<td>ruVertical = 2</td>
<td>Only vertical line</td>
</tr>
<tr>
<td>ruHorizontal = 3</td>
<td>Only horizontal line</td>
</tr>
</tbody>
</table>

See Also \texttt{CursorPos} property and \texttt{TxRulerType} constants.

4.1.3.1.2.15 StyleName

Sets/gets the Style from the collection of styles saved through the \texttt{TNBXProfiles} ActiveX Control.

**Visual Basic Syntax**

\texttt{TNBXEmulator.StyleName} [= \texttt{String}]

**Delphi Syntax**

\texttt{TNBXEmulator.StyleName} [:= \texttt{String}];

**Remarks**

This property is valid if a \texttt{ProfilesControl} property has been previously assigned.

See Also \texttt{TNBXProfiles} control and \texttt{ShowEditor} method.

4.1.3.1.2.16 SubKey

Sets/gets the subkey under which the screen styles will be stored.

**Visual Basic Syntax**

\texttt{TNBXEmulator.SubKey} [= \texttt{String}]

**Delphi Syntax**

\texttt{TNBXEmulator.SubKey} [:= \texttt{String}];
**Remarks**

This property is valid only if a ProfilesControl property has been assigned to a TNBXProfiles control.

See also ProfilesControl property, SubKey and TNBXProfiles Control.

### 4.1.3.1.2.17 TelnetControl

Sets the TNB3270X or TNB5250X ActiveX Control as the telnet client control.

**Visual Basic Syntax**

```vbnet
TNBXEmulator.TelnetControl [= TNBnnnnX]
```

**Delphi Syntax**

```delphi
TNBXEmulator.TelnetControl [:= TNBnnnnX];
```

**Remarks**

You must set this property for TNBEmulator control works properly.

### 4.1.3.1.2.18 TopMargin

Sets emulation screen top margin.

**Visual Basic Syntax**

```vbnet
TNBXEmulator.TopMargin [= Integer]
```

**Delphi Syntax**

```delphi
TNBXEmulator.TopMargin [:= Integer];
```

**Remarks**

Top margin emulation value is measured in pixels.

### 4.1.3.1.2.19 Visible

Sets/gets the visibility of the control at run-time.

**Visual Basic Syntax**

```vbnet
TNBXEmulator.Visible [= Boolean]
```
**Delphi Syntax**

```
TNBXEmulator.Visible [:= Boolean];
```

**Remarks**

Default value is True.

### 4.1.3.1.3 Methods

#### 4.1.3.1.3.1 AboutBox

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```
TNBXEmulator.AboutBox
```

**Delphi Syntax**

```
TNBXEmulator.AboutBox;
```

#### 4.1.3.1.3.2 AidKeyPressed

AidKeyPressed can be used to send AID keys to the host.

**Visual Basic Syntax**

```
TNBXEmulator.AidKeyPressed (Value As String)
```

**Delphi Syntax**

```
TNBXEmulator.AidKeyPressed (Value:string);
```

**Remarks**

The AID key go directly to the mainframe's host application. Doesn't remains local.

See also **AIDKey** and **LocalKeyPressed** properties, **SendKeys** and **SendAid** methods, and **OnAidKey** event.

#### 4.1.3.1.3.3 AsVariant

Returns the control as a variant data type variable.

**Visual Basic Syntax**

```
[Variant =] TNBXEmulator.AsVariant
```
4.1.3.1.3.4 ClassNameIs
Indicates if a specified name is the corresponding class name.

**Visual Basic Syntax**

[Boolean =] TNBXEmulator.ClassNameIs (Name As String)

**Delphi Syntax**

[Boolean :=] TNBXEmulator.ClassNameIs (Name: string);

4.1.3.1.3.5 Copy
The Copy method copies the selected area to the clipboard.

**Visual Basic Syntax**

TNBXEmulator.Copy

**Delphi Syntax**

TNBXEmulator.Copy;

**Remarks**

The EnableSelection property must be set to true to allows you select an area of the emulator-screen.

See also EnableSelection property and Paste method.

4.1.3.1.3.6 GetScreenRowEx
GetScreenRowEx can be used to retrieve the text buffer of the one row of the current screen. You can choose to recover the attribute or extended attribute of each field together to the character data.

**Visual Basic Syntax**

[String =] TNBXEmulator.GetScreenRowEx (Row As Integer, [Attr As Boolean = True], [Eab As Boolean = True])

**Delphi Syntax**

[String :=] TNBXEmulator.GetScreenRowEx (Row:Integer, [Attr:Boolean = True],
**Remarks**

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in Row. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for fields attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

See also [GetScreenText](#) method.

### 4.1.3.1.3.7 GetScreenText

GetScreenText returns the text buffer of a portion of the current screen. You can choose to recover the attribute or extended attribute of each field together to the character data.

**Visual Basic Syntax**

```vbnet
[String] = TNBXEmulator.GetScreenText (StartPos As Integer, EndPos As Integer, [Attr As Boolean = True], [Eab As Boolean = True])
```

**Delphi Syntax**

```delphi
[String :]= TNBXEmulator.GetScreenText (StartPos:Integer, EndPos:Integer, 
[Attr:Boolean = True], [Eab:Boolean = True]);
```

**Remarks**

StartPos and EndPos specify the start position and the end position of the text buffer that will be returned.

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in Row. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for fields attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.
See also GetScreenWidthEx method.

4.1.3.1.3.8 LoadFromXMLFile

Loads the XML code corresponding to the TNBXEmulator object.

**Visual Basic Syntax**

```vb
TNBXEmulator.LoadFromXMLFile (XMLFile As String)
```

**Delphi Syntax**

```delphi
TNBXEmulator.LoadFromXMLFile (XMLFile: string);
```

See also SaveToXMLFile method.

4.1.3.1.3.9 LocalKeyPressed

LocalKeyPressed can be used to send AID keys to be executed in local mode.

**Visual Basic Syntax**

```vb
TNBXEmulator.LocalKeyPressed (Value As String)
```

**Delphi Syntax**

```delphi
TNBXEmulator.LocalKeyPressed (Value: String);
```

**Remarks**

The AID key doesn't go to the mainframe's host application. Remains local.

See also AIDKey property, AidKeyPressed, SendKeys and SendAid methods and OnAidKey event.

4.1.3.1.3.10 Paste

The Paste method pastes text from the clipboard to the current cursor position.

**Visual Basic Syntax**

```vb
TNBXEmulator.Paste
```

**Delphi Syntax**

```delphi
TNBXEmulator.Paste;
```

See also Copy method.
4.1.3.1.3.11 PrintScreen

Prints all the emulator-screen area.

Visual Basic Syntax

TNBXEmulator.PrintScreen

Delphi Syntax

TNBXEmulator.PrintScreen;

4.1.3.1.3.12 PutFields

Sends the modified fields to the TelnetControl in use by TNBEmulator control.

Visual Basic Syntax

TNBXEmulator.PutFields

Delphi Syntax

TNBXEmulator.PutFields;

Remarks

To complete the data transmission to the host, you must set the AidKey property or use the SendAid method from the TelnetControl Control.

See also TelnetControl property from TNBEmulator control, and, AidKey property and SendAid method from TNB3270X/TNB5250X controls.

4.1.3.1.3.13 SaveToXMLFile

Exports the XML code corresponding to the TNBXEmulator object.

Visual Basic Syntax

TNBXEmulator.SaveToXMLFile (XMLFile As String)

Delphi Syntax

TNBXEmulator.SaveToXMLFile (XMLFile: string);

See also LoadFromXMLFile method.

4.1.3.1.3.14 SendKeys

SendKeys can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host
functions.

**Visual Basic Syntax**

`TNBXEmulator.SendKeys (Keys As String)`

**Delphi Syntax**

`TNBXEmulator.SendKeys (Keys: string);`

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the `Keys` parameter:

"logon applid(cics)@E"
Remarks
The execution of the SendKeys is stopped when an AID Key is sent to the Host.

See also **AIDKey** property and **SendKeys** method.

4.1.3.1.3.15 ShowEditor

Shows a modal dialog with the screen properties.

**Visual Basic Syntax**

`TNBXEmulator.ShowEditor`

**Delphi Syntax**

`TNBXEmulator.ShowEditor;`

Remarks
If you haven't assigned the **ProfilesControl** property, the style list will be shown disabled. Setting the **ProfilesControl** property to a valid **TNBXProfiles** control, allows you to define screen-styles at run-time and save them into a file for future reuse.

See also **TNBXProfiles** property.

4.1.3.1.4 Events

4.1.3.1.4.1 OnClick

Occurs when the user clicks the control.

See also **OnDblClick** event.

4.1.3.1.4.2 OnCursorPos

Occurs when the emulator cursor changes its position.

4.1.3.1.4.3 OnDblClick

Occurs when the user double-clicks the primary mouse button when the mouse pointer is over the control.

See also **OnClick** event.

4.1.3.1.4.4 OnInputInhibitChange

Occurs when the input from keyboard changes to inhibit or vice versa.

See also **OnInsertModeChange** event.
4.1.3.1.4.5 OnInsertModeChange

Occurs when the input from keyboard changes to insert mode or overwrite mode.

See also OnInputInhibitChange event.

4.1.3.1.5 Constants

4.1.3.1.5.1 TxRulerType

These values are used in the RulerType property of TNBXEmulator control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ruNone = 0</td>
<td>No lines</td>
</tr>
<tr>
<td>ruCrosshair = 1</td>
<td>Horizontal and vertical lines</td>
</tr>
<tr>
<td>ruVertical = 2</td>
<td>Only vertical line</td>
</tr>
<tr>
<td>ruHorizontal = 3</td>
<td>Only horizontal line</td>
</tr>
</tbody>
</table>

4.1.3.2 TNBXAidHook Control

4.1.3.2.1 TNBXAidHook Control

This is an implementation of a keyboard interface, with mapping to the standard function keys of the 5250 and 3270 emulation.

**Properties**

- Active
- HWND
- KbdFile
- KbdKind
- Mapping3270
- Mapping5250
- Profile
- ProfilesControl
- SubKey

**Methods**

- AboutBox
- AsVariant
- LoadFromXMLFile
- SaveToXMLFile
- ShowEditor

**Events**

- OnAidKey
4.1.3.2.2 Properties

4.1.3.2.2.1 Active

Enables/disables the keyboard hook.

**Visual Basic Syntax**

TNBXAidHook.**Active** [= Boolean]

**Delphi Syntax**

TNBXAidHook.**Active** [:= Boolean];

**Remarks**

Default value is **False**.

4.1.3.2.2.2 HWND

Sets/gets the handle to the window which the hook is being applied.

**Visual Basic Syntax**

TNBXAidHook.**HWND** [= Integer]

**Delphi Syntax**

TNBXAidHook.**HWND** [:= Integer];

**Remarks**

You can set the Keyboard Hook to be applied to any window. The OnAidKey event handler will be fired when a combination of keys in the mapping table is pressed.

See also **OnAidKey** event.

4.1.3.2.2.3 KbdFile

Allows you to specify an external file with a new keyboard mapping.

**Visual Basic Syntax**

TNBXAidHook.**KbdFile** [= String]

**Delphi Syntax**

TNBXAidHook.**KbdFile** [:= String];

**Remarks**
The file to be used can be exported from its property page or dialog box shown using the `ShowEditor` method.

See Also `ShowEditor` method.

### 4.1.3.2.2.4 KbdKind

Specify the keyboard mapping kind.

**Visual Basic Syntax**

```vb
TNBXAidHook.KbdKind [= TNBKbdKind]
```

**Delphi Syntax**

```delphi
TNBXAidHook.KbdKind [:= TNBKbdKind];
```

It can take one of this constants values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>k5250</td>
<td>5250 Keyboard</td>
</tr>
<tr>
<td>k3270</td>
<td>3270 Keyboard</td>
</tr>
</tbody>
</table>

**Remarks**

This control maintain separates keyboard mapping tables for each connection type. You can modify it through the property page associated to the control.

See Also `TNBKbdKind` constants.

### 4.1.3.2.2.5 Mapping3270

Shows 3270 keyboard mapping dialog box for creating, editing and deleting keyboard maps.

**Visual Basic Syntax**

```vb
TNBXAidHook.Mapping3270 [= KeyboardMap]
```

**Delphi Syntax**

```delphi
TNBXAidHook.Mapping3270 [:= KeyboardMap];
```

See Also `Mapping5250` property.

### 4.1.3.2.2.6 Mapping5250

Shows 5250 keyboard mapping dialog box for creating, editing and deleting keyboard maps.

**Visual Basic Syntax**
**TNBXAidHook.** `Mapping5250` \[= KeyboardMap\]

### Delphi Syntax

`TNBXAidHook. Mapping5250 := KeyboardMap;`

See Also [Mapping3270](#) property.

#### 4.1.3.2.2.7 Profile

Sets or gets the name of the current keyboard map set.

**Visual Basic Syntax**

`TNBXAidHook. Profile \[= String\]`

**Delphi Syntax**

`TNBXAidHook. Profile \[:= String\];`

### Remarks

The default keyboard map is called "Default". You can access a collection of keyboard maps to be generated and customized through the `ShowEditor` method. Then you can assign a different profile using `Profile` property.

Different profiles generated will be stored through `TNBXProfiles` control and the physical file will be identified through `KbdFile` property.

See Also `TNBXProfiles` control, `SubKey` and `KbdKind` properties and `ShowEditor` method.

#### 4.1.3.2.2.8 ProfilesControl

Sets the `TNBXProfiles` ActiveX Control as the profile manager for this control.

**Visual Basic Syntax**

`TNBXAidHook. ProfilesControl \[= TNBXProfiles\]`

**Delphi Syntax**

`TNBXAidHook. ProfilesControl \[:= TNBXProfiles\];`

### Remarks

After setting this property, you can access a collection of keyboard maps to be generated and customized through the `ShowEditor` method.

See Also `TNBXProfiles` control, `SubKey` property and `ShowEditor` method.
4.1.3.2.2.9 SubKey

Sets/gets the subkey under which the keyboard maps will be stored.

**Visual Basic Syntax**

\[TNBXAidHook.SubKey \[=\] String\]

**Delphi Syntax**

\[TNBXAidHook.SubKey \[:=\] String;\]

**Remarks**

This property is valid only if a ProfilesControl property has been assigned to a TNBXProfiles control.

See also TNBXProfiles control, ProfilesControl property and ShowEditor method.

4.1.3.2.3 Methods

4.1.3.2.3.1 AboutBox

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

\[TNBXAidHook.AboutBox\]

**Delphi Syntax**

\[TNBXAidHook.AboutBox;\]

4.1.3.2.3.2 AsVariant

Returns the control as a variant data type variable.

**Visual Basic Syntax**

\[Variant =] TNBXAidHook.AsVariant\]

**Delphi Syntax**

\[Variant :=] TNBXAidHook.AsVariant;\]

4.1.3.2.3.3 GetControlHandle

Retrieves an internal control handle for this control.

**Visual Basic Syntax**

\[Integer =] TNBXAidHook.GetControlHandle\]
Delphi Syntax

[Integer :=] TNBXAidHook.GetControlHandle;

Remarks

This handle is to be used in others Integration Pack ActiveX Controls.

See also AidHookControl property from TNBXEmulator control.

4.1.3.2.3.4 LoadFromXMLFile

Loads the XML code corresponding to the TNBXAidHook object.

Visual Basic Syntax

TNBXAidHook.LoadFromXMLFile (XMLFile As String)

Delphi Syntax

TNBXAidHook.LoadFromXMLFile (XMLFile: string);

See Also SaveToXMLFile method.

4.1.3.2.3.5 SaveToXMLFile

Exports the XML code corresponding to the TNBXAidHook object.

Visual Basic Syntax

TNBXAidHook.SaveToXMLFile (XMLFile as String)

Delphi Syntax

TNBXAidHook.SaveToXMLFile (XMLFile:string);

See also LoadFromXMLFile method.

4.1.3.2.3.6 ShowEditor

Shows a modal dialog with the screen properties.

Visual Basic Syntax

TNBXAidHook.ShowEditor

Delphi Syntax

TNBXAidHook.ShowEditor;
4.1.3.2.4 Events

4.1.3.2.4.1 OnAidKey

Occurs when a combination of keys in the mapping table is pressed.

**Visual Basic Syntax**

```vbnet
TNBXAidHook_OnAidKey (ByVal Value As String)
```

**Delphi Syntax**

```delphi
TNBXAidHook.OnAidKey (Sender: TObject)
```

**Remarks**

Value parameter can be one of the following strings:

Local processing function keys:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up</td>
<td>go to previous row</td>
</tr>
<tr>
<td>Down</td>
<td>go to next row</td>
</tr>
<tr>
<td>Left</td>
<td>go to previous column</td>
</tr>
<tr>
<td>Right</td>
<td>go to next column</td>
</tr>
<tr>
<td>Insert</td>
<td>set the keyboard in inserting state</td>
</tr>
<tr>
<td>Delete</td>
<td>delete the next character</td>
</tr>
<tr>
<td>BackSpace</td>
<td>delete the previous character</td>
</tr>
<tr>
<td>ScreenBegin</td>
<td>go to row 1 column 1</td>
</tr>
<tr>
<td>ScreenEnd</td>
<td>go to last row last column</td>
</tr>
<tr>
<td>NextLine</td>
<td>go to column 1 of the next line</td>
</tr>
<tr>
<td>PriorField</td>
<td>go to the previous field</td>
</tr>
<tr>
<td>NextField</td>
<td>go to the next field</td>
</tr>
<tr>
<td>Home</td>
<td>go to the first field</td>
</tr>
<tr>
<td>End</td>
<td>go to last character of current field</td>
</tr>
<tr>
<td>EraseField</td>
<td>erase the current field</td>
</tr>
<tr>
<td>EraseEof</td>
<td>erase characters from current position until the end of field</td>
</tr>
<tr>
<td>FieldMinus</td>
<td>skip to the previous field</td>
</tr>
<tr>
<td>FieldPlus</td>
<td>skip to the next field</td>
</tr>
<tr>
<td>Restore</td>
<td>restore the current screen</td>
</tr>
</tbody>
</table>

Host processing function keys (Attention Identifier "AID" keys):

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
</tbody>
</table>
### Component Reference for ActiveX and Delphi

<table>
<thead>
<tr>
<th>SysReq</th>
<th>System Request key</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>PF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>PF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>PF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>PF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>PF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>PA1</td>
<td>PA1 key</td>
</tr>
</tbody>
</table>

See Also [AIDKey](#) property, [AidKeyPressed](#), [LocalKeyPressed](#), [SendKeys](#) and [SendAid](#) methods.

### 4.1.3.2.5 Constants

#### 4.1.3.2.5.1 TNBKbdKind

These values are used in the [KbdKind property](#) of telnet controls.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>k5250</td>
<td>5250 Keyboard</td>
</tr>
<tr>
<td>k3270</td>
<td>3270 Keyboard</td>
</tr>
</tbody>
</table>

### 4.1.3.3 TNBXStatusBar Control

#### 4.1.3.3.1 TNBXStatusBar Control

This control implements a status bar that shows the connection and session status.

**Properties**

- [EmulatorControl](#)
- [Font](#)
- [TelnetControl](#)
• **Visible**

**Methods**

• **AboutBox**
• **AsVariant**

4.1.3.3.2 Properties

4.1.3.3.2.1 EmulatorControl

Sets the TNBXEmulator ActiveX Control for emulator events handling.

**Visual Basic Syntax**

```vbnet
TNBStatusBar.EmulatorControl [= TNBEmulator]
```

**Delphi Syntax**

```delphi
TNBStatusBar.EmulatorControl [:= TNBEmulator];
```

**Remarks**

You must set this property for TNBXStatusBar control to work properly.

4.1.3.3.2.2 Font

Sets/gets the font for the current connection.

**Visual Basic Syntax**

```vbnet
TNBStatusBar.Font [= IFontDisp]
```

**Delphi Syntax**

```delphi
TNBStatusBar.Font [:= IFontDisp];
```

4.1.3.3.2.3 TelnetControl

Sets the TNB3270X or TNB5250X ActiveX Control as the telnet client control.

**Visual Basic Syntax**

```vbnet
TNBStatusBar.TelnetControl [= TNBnnnnX]
```

**Delphi Syntax**

```delphi
TNBStatusBar.TelnetControl [:= TNBnnnnX];
```

**Remarks**

You must set this property for TNBStatusBar control to allow it to work properly.
4.1.3.2.4 Visible

Sets/gets the visibility of the control at run-time.

**Visual Basic Syntax**

TNBXStatusBar. Visible [ = Boolean]

**Delphi Syntax**

TNBXStatusBar. Visible [ := Boolean];

**Remarks**

Default value is True.

4.1.3.3 Methods

4.1.3.3.1 AboutBox

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

TNBXStatusBar. AboutBox

**Delphi Syntax**

TNBXStatusBar. AboutBox;

4.1.3.3.2 AsVariant

Returns the control as a variant data type variable.

**Visual Basic Syntax**

[Variant =] TNBXStatusBar. AsVariant

**Delphi Syntax**

[Variant :=] TNBXStatusBar. AsVariant;

4.1.3.4 TNBXMacro Control

4.1.3.4.1 TNBXMacro Control

This control implements a way to record and play send/receive screen sequences. In a sequence you have fields list, cursor positions, Aid keys, etc.

**Properties**

- AutoStartMacro
- MacroName
- ProfilesControl
• State
• SubKey
• TelnetControl

Methods

• AboutBox
• AsVariant
• LoadFromXMLFile
• SaveToXMLFile
• StartPlaying
• StartRecording
• StopPlaying
• StopRecording

Events

• OnStartPlaying
• OnStartRecording
• OnStopPlaying
• OnStopRecording

4.1.3.4.2 Properties

4.1.3.4.2.1 AutoStartMacro

Sets/gets the macro name that will be auto started when the connection has been successfully established.

Visual Basic Syntax

TNBXMacro.AutoStartMacro [= String]

Delphi Syntax

TNBXMacro.AutoStartMacro [:= String];

Remarks

If you have assigned the ProfilesControl property to a TNBXProfiles control, this property refers to a macro stored in the file pointed by this control; otherwise you must specify a valid filename of a macro file previously saved.

See Also ProfilesControl property, StopRecording and StartPlaying methods.

4.1.3.4.2.2 MacroName

Returns the name of the macro loaded. Sets the macro to be loaded from the ProfilesControl or a file. You can have one file for each macro, or one file containing several macros.

Visual Basic Syntax
`TNBXMacro.MacroName` \[= String\]

**Delphi Syntax**

```delphi
TNBXMacro.MacroName [:= String];
```

**Remarks**

If you have assigned the ProfilesControl property to a TNBXProfiles control, you can retrieve a specific macro by its name using this property. Otherwise, the MacroName property must specify a valid filename containing a previously saved macro.

See Also `ProfilesControl` property, `StopRecording` and `StartPlaying` methods.

### 4.1.3.4.2.3 ProfilesControl

Sets the `TNBXProfiles` ActiveX Control as the profile manager for this control.

**Visual Basic Syntax**

```vbnet
TNBXMacro.ProfilesControl [= `TNBXProfiles`]
```

**Delphi Syntax**

```delphi
TNBXMacro.ProfilesControl [:= `TNBXProfiles`];
```

**Remarks**

After setting this property, you can save and load macros from the file pointed in `TNBXProfiles` control. You can retrieve a specific macro by its name using the `MacroName` property.

See Also `TNBXProfiles` control, `SubKey` and `MacroName` properties.

### 4.1.3.4.2.4 State

Returns the state of the control.

**Visual Basic Syntax**

```vbnet
[TNBMacroState ] = TNBXMacro.State
```

**Delphi Syntax**

```delphi
[TNBMacroState ] := TNBXMacro.State;
```

**Remarks**
The value of this property can be one of the following constants:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>msNoMacro</td>
<td>No macro is loaded.</td>
</tr>
<tr>
<td>msStopped</td>
<td>The control has a macro loaded and is in a stopped state.</td>
</tr>
<tr>
<td>msRecording</td>
<td>The control is recording a macro.</td>
</tr>
<tr>
<td>msPlaying</td>
<td>The control is playing the macro loaded.</td>
</tr>
</tbody>
</table>

See Also ProfilesControl property, StopRecording and StartPlaying methods.

4.1.3.4.2.5 SubKey

Sets/gets the subkey under which will be stored the macro sequences.

**Visual Basic Syntax**

TNBXMacro.SubKey [= String]

**Delphi Syntax**

TNBXMacro.SubKey := String;

**Remarks**

This property is valid only if a ProfilesControl property has been assigned to a TNBXProfiles control.

See also ProfilesControl property and TNBXProfiles Control.

4.1.3.4.2.6 TelnetControl

Sets the TNB3270X or TNB5250X ActiveX Control as the telnet client control.

**Visual Basic Syntax**

TNBXMacro.TelnetControl [= TNBnnnnX]

**Delphi Syntax**

TNBXMacro.TelnetControl := TNBnnnnX;

**Remarks**

You must set this property for TNBXMacro control to allow it to work properly.

4.1.3.4.3 Methods

4.1.3.4.3.1 AboutBox

Shows Integration Pack about dialog box.
Visual Basic Syntax

`TNBXMacro.AboutBox`

Delphi Syntax

`TNBXMacro.AboutBox;`

4.1.3.4.3.2  AsVariant

Returns the control as a variant data type variable.

Visual Basic Syntax

`[Variant =] TNBXMacro.AsVariant`

Delphi Syntax

`[Variant :=] TNBXMacro.AsVariant;`

4.1.3.4.3.3  LoadFromXMLFile

Loads the XML code corresponding to the `TNBXMacro object`.

Visual Basic Syntax

`TNBXMacro.LoadFromXMLFile (FileName As String)`

Delphi Syntax

`TNBXMacro.LoadFromXMLFile (FileName: TFileName);`

See also `SaveToXMLFile` method.

4.1.3.4.3.4  SaveToXMLFile

Exports the XML code corresponding to the `TNBXMacro object`.

Visual Basic Syntax

`TNBXMacro.SaveToXMLFile (FileName As TFileName)`

Delphi Syntax

`TNBXMacro.SaveToXMLFile (FileName: TFileName);`

See also `LoadFromXMLFile` method.

4.1.3.4.3.5  StartPlaying

Starts playing a macro.
Visual Basic Syntax

TNBXMacro.StartPlaying (Name as String)

Delphi Syntax

TNBXMacro.StartPlaying (Name:string);

Remarks

Starts playing the macro specified in the Name parameter. The Name parameter follow the same rules as MacroName property.

See also MacroName property, StopPlaying method and OnStartPlaying event.

4.1.3.4.3.6 StartRecording

Starts the macro recording process.

Visual Basic Syntax

TNBXMacro.StartRecording()

Delphi Syntax

TNBXMacro.StartRecording();

Remarks

Starts recording send and receive sequences. The process stops with the StopRecording method.

See also StopRecording method and OnStartRecording event.

4.1.3.4.3.7 StopPlaying

Stops playing the macro.

Visual Basic Syntax

TNBXMacro.StopPlaying()

Delphi Syntax

TNBXMacro.StopPlaying();

Remarks

Stops playing the current macro. Normally the macro stops when all its steps were successfully executed. However, in same cases the macro doesn't stop automatically
because of changes on the screen sequences or field positions.

See also StartPlaying method OnStopPlaying event.

4.1.3.4.3.8 StopRecording

Stops the macro recording process.

**Visual Basic Syntax**

```
TNBXMacro.StopRecording (Name as String)
```

**Delphi Syntax**

```
TNBXMacro.StopRecording (Name:string);
```

**Remarks**

Stops the macro recording process and save the macro in a file. The Name parameter follow the same rules of the MacroName property.

See also MacroName property, StartRecording, StartPlaying methods and OnStopRecording event.

4.1.3.4.4 Events

4.1.3.4.4.1 OnStartPlaying

The event is fired when the playing process starts.

See also StartPlaying method.

4.1.3.4.4.2 OnStartRecording

The event is fired when the recording process starts.

See also StartRecording method.

4.1.3.4.4.3 OnStopPlaying

The event is fired when the playing process stops.

See also StopPlaying method.

4.1.3.4.4.4 OnStopRecording

The event is fired when the recording process stops.

See also StopRecording method.
4.1.3.4.5 Constants

4.1.3.4.5.1 TNBMacroState

These constants are used by the State property and let you know the current state of the control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>msNoMacro</td>
<td>No macro is loaded.</td>
</tr>
<tr>
<td>msStopped</td>
<td>The control has a macro loaded and is in a stopped state.</td>
</tr>
<tr>
<td>msRecording</td>
<td>The control is recording a macro.</td>
</tr>
<tr>
<td>msPlaying</td>
<td>The control is playing the macro loaded.</td>
</tr>
</tbody>
</table>

4.1.3.5 TNBXProfiles Control

4.1.3.5.1 TNBXProfiles Control

This control implements a way to store profiles in an OLE Compound Document file or an XML file.

Properties

- **FileName**
- **Items**
- **Key**
- **ReadOnly**
- **StreamingType**

Methods

- **AboutBox**
- **AsVariant**
- **Commit**
- **Delete**
- **GetCurrent**
- **GetXMLHeader**
- **Rename**
- **Rollback**
- **setCurrent**

4.1.3.5.2 Properties

4.1.3.5.2.1 FileName

Sets or gets the file name for a profile. This file will be used to save a profile or to load an existing one.

**Visual Basic Syntax**

```vbnet
TNBXProfiles.FileName [ = String ]
```

**Delphi Syntax**
TNBXProfiles.FileName := String;

Remarks

Internally a profile file is divided in sections and subsections (keys and subkeys). You can use only one file for all kind of profiles you want to save, for example: connections, keyboard maps, macros, screen styles and hotspots, or one file for each kind of profile.

Sections can be accessed through Key property and subsections through SubKey property.

If you don't specify a directory, the file must be in the OCX's directory (the directory where the component is installed). We recommend the use of the .tbp extension to identify a profile file (for example: hollis.tbp).

See Also Key property, SubKey property of TNB5250X/TNB3270X controls, SubKey property of TNBXEmulator control, SubKey property of TNBXAidHook control, SubKey property of TNBXMacro control and SubKey property TNBXHotSpots collection.

4.1.3.5.2.2 Items

Returns a String list containing the profiles stored under the Key property and the Subkey parameter. The values of Key property and Subkey parameter are in the file name specified in FileName property.

Visual Basic Syntax

[OleVariant =] TNBXProfiles.Items (SubKey As String)

Delphi Syntax

[OleVariant :=] TNBXProfiles.Items [SubKey:string];

Remarks

You can use the Items.Count property to know the total number of items in the list.

See Also FileName and Key properties.

4.1.3.5.2.3 Key

Sets or gets the key that will be used to store the profile.

Visual Basic Syntax

TNBXProfiles.Key [= String]

Delphi Syntax

TNBXProfiles-Key [= String];
Remarks

If you share the file between two or more TNBXProfiles controls, the Key property must be different in order to store the profiles in different paths.

See Also FileName property.

4.1.3.5.2.4 ReadOnly

Indicates if the profile file has read only attribute set.

Visual Basic Syntax

[Boolean =] TNBXProfiles.ReadOnly

Delphi Syntax

[Boolean :=] TNBXProfiles.ReadOnly;

Remarks

If the attribute means that the profile file is read-only (ReadOnly property returning True value), profile information will not be saved preventing an error message to occur.

See Also FileName property.

4.1.3.5.2.5 StreamingType

Sets/gets the profile's streaming type.

Visual Basic Syntax

TNBXProfiles.StreamingType [= TStreamingType]

Delphi Syntax

TNBXProfiles.StreamingType [= TStreamingType];

Remarks

TStreamingType can be either XML or OleDoc format. This format will be used to read and write profiles.

4.1.3.5.3 Methods

4.1.3.5.3.1 AboutBox

Shows Integration Pack about dialog box.
Visual Basic Syntax

`TNBXProfiles.AboutBox`

Delphi Syntax

`TNBXProfiles.AboutBox;`

4.1.3.5.3.2 AsVariant

Returns the control as a variant data type variable.

**Visual Basic Syntax**

```vbnet
[Variant = ] TNBXProfiles.AsVariant
```

**Delphi Syntax**

```delphi
[Variant := ] TNBXProfiles.AsVariant;
```

4.1.3.5.3.3 Commit

Writes the XML configuration to the profile.

**Visual Basic Syntax**

```vbnet
TNBXProfiles.Commit
```

**Delphi Syntax**

```delphi
TNBXProfiles.Commit;
```

See also **Rollback** method.

4.1.3.5.3.4 Delete

Deletes the profile specified.

**Visual Basic Syntax**

```vbnet
TNBXProfiles.Delete (SubKey As String, Profile As String)
```

**Delphi Syntax**

```delphi
TNBXProfiles.Delete (SubKey:string, Profile:string);
```

**Remarks**

SubKey parameter is the subkey of the caller control and Profile parameter is the name of the configuration that is going to be deleted.
See also `Key` and `FileName` properties, and `Rename` method.

### 4.1.3.5.3.5 GetCurrent

Returns a string with the current profile.

**Visual Basic Syntax**

```
[Profile =] TNBXProfiles.GetCurrent (SubKey As String)
```

**Delphi Syntax**

```
[Profile :=] TNBXProfiles.GetCurrent (SubKey As String);
```

**Remarks**

This method is generally used to get the default profile name to be loaded.

### 4.1.3.5.3.6 GetXMLHeader

Gets the XML Header with the Name, Tag, Id and Type attributes.

**Visual Basic Syntax**

```
TNBXProfiles.GetXMLHeader (Tag As String, Name As String, Id As String, Type_ As String)

TNBXProfiles.GetXMLHeader (Filename As String, Tag As String, Name As String, Id As String, Type_ As String)
```

**Delphi Syntax**

```
TNBXProfiles.GetXMLHeader (var Tag, Name, Id, Type_: string);

TNBXProfiles.GetXMLHeader (Filename: string; var Tag, Name, Id, Type_: string);
```

### 4.1.3.5.3.7 Rename

Renames the profile specified.

**Visual Basic Syntax**

```
TNBXProfiles.Rename (Source As String, OldProfile As String, Profile As String, Id As String= '', type_ As string = '');
```
**Delphi Syntax**

```delphi
TNBXProfiles.Rename (Source:string;OldProfile, Profile: string;Id:string= ''; type_: string = '');
```

**Remarks**

OldProfile parameter is the name of the profile that is going to be renamed and NewProfile parameter is the new name of the profile.

See also `Key` and `FileName` properties, and `Delete` method.

### 4.1.3.5.3.8 Rollback

Rollbacks changes done to the XML configuration.

**Visual Basic Syntax**

```vbnet
TNBXProfiles.Rollback
```

**Delphi Syntax**

```delphi
TNBXProfiles.Rollback;
```

See also `Commit` method.

### 4.1.3.5.3.9 SetCurrent

Sets the current profile.

**Visual Basic Syntax**

```vbnet
TNBXProfiles.SetCurrent (Source As String, CurrentProfile As String, id As String=' ', type_ As string=' ')```

**Delphi Syntax**

```delphi
TNBXProfiles.SetCurrent (Source:string;CurrentProfile: string; id:string=' ';type_: string=' ')
```

**Remarks**

This method is generally used to set the default profile name to be loaded the next time the application starts.

### 4.1.3.6 TNBXHotSpots Class Reference

#### 4.1.3.6.1 TNBXHotSpots Class Reference

HotSpots allows you transform screen text associated with terminal emulation tasks, to active elements such as buttons, links, etc.
These active elements are defined through a pattern (sample text or regular expression) and will execute keystroke sequences through a simple mouse click. Also HotSpots are delimited to a specific screen area by row-col coordinates. The primary objective of a hotspot is to replace aid keys in a graphic form, and are direct actions of commands described in the host screen.

Properties

- **ActionFieldData**
- **Active**
- **Background**
- **Description**
- **EndCol**
- **EndRow**
- **Id**
- **Pattern**
- **StartCol**
- **StartRow**
- **TextColor**
- **ViewAs**

Constants

- **TXTNbHotSpotViewAs**

4.1.3.6.2 Properties

4.1.3.6.2.1 ActionFieldData

Sets or gets the data to be sent to internal SendKeys method. Here you can specify key sequences to the mainframe, including AID keys. The syntax to be used for the data to be sent to the mainframe, is the same that uses SendKeys method of TNB5250/TNB3270/TNBSync Controls.

**Visual Basic Syntax**

```vbnet
TNBXHotSpot.ActionFieldData [= String]
```

**Delphi Syntax**

```delphi
TNBXHotSpot.ActionFieldData [:= String];
```

**Remarks**

For a complete guide of how to complete the string to be assigned to this property, see table lists of functions keys and its corresponding codes in SendKeys method of TNBX5250/TNBX3270 Controls or SendKeys method of TNBXSync control.

A simple example:

```plaintext
TNBXHotSpot1.ActionFieldData="L@E"
```
See Also **SendKeys** method (TNBX5250/TNB3270), **SendKeys** method (TNBXSync).

### 4.1.3.6.2.2 Active

Enables/disables the hotspot.

**Visual Basic Syntax**

```
TNBXHotSpot.Active [= Boolean]
```

**Delphi Syntax**

```
TNBXHotSpot.Active [= Boolean];
```

**Remarks**

In order to function properly, hotspot control must be active and also link with **TNBXEmulator** control.

Default value for this property is **True**.

See Also **TNBXEmulator** control, **HotSpotControl** property.

### 4.1.3.6.2.3 Background

Sets the hotspot background color to be shown in a host screen.

**Visual Basic Syntax**

```
TNBXHotSpot.Background [= OLE_COLOR]
```

**Delphi Syntax**

```
TNBXHotSpot.Background [= OLE_COLOR];
```

**Remarks**

This property returns a long value that represent a hotspot background color.

See Also **TextColor** and **ViewAs** properties.

### 4.1.3.6.2.4 Description

Sets/gets the description of the hotspot. This is simply a descriptive string associated with a hotspot.

**Visual Basic Syntax**

```
TNBXHotSpot.Description [= String]
```

**Delphi Syntax**
 TNBXHotSpot.Description [:: String];

See Also Id property.

4.1.3.6.2.5 EndCol
Sets/gets the column where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

**Visual Basic Syntax**

TNBXHotSpot.EndCol [:= Integer]

**Delphi Syntax**

TNBXHotSpot.EndCol [:= Integer];

**Remarks**
Valid values are from 1 to TNBX5250/TNBX3270.Cols property.

See Also EndRow, StartCol, StartRow properties.

4.1.3.6.2.6 EndRow
Sets/gets the row where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

**Visual Basic Syntax**

TNBXHotSpot.EndRow [= Integer]

**Delphi Syntax**

TNBXHotSpot.EndRow [:= Integer];

**Remarks**
Valid values are from 1 to TNBX5250/TNBX3270.Rows property.

See Also StartRow, EndCol, StartCol properties.
4.1.3.6.2.7 Id

Unique Id of the hotspot. This is only for internal use.

**Visual Basic Syntax**

```
TNBXHotSpot.Id := String
```

**Delphi Syntax**

```
TNBXHotSpot.Id := String;
```

See Also [Description](#) property.

4.1.3.6.2.8 Pattern

Sets/gets a regular host screen search expression for hotspot recognition and its graphical draw. When a hotspot is set in your application design time, it needs a search expression where Integration Pack can identify and display the hotspot at application run time.

**Visual Basic Syntax**

```
TNBXHotSpot.Pattern := String
```

**Delphi Syntax**

```
TNBXHotSpot.Pattern := string;
```

**Remarks**

This property is a string value that represent part or all the text displayed of a host screen field where you want to display a hotspot instead of the original text.

A simple example:

Suppose that a host screen has a field that denotes how you can access a specific application option, for example, accessing a public library with the command L. The text displayed in the host field will be like this:

```
L - LIBRARY (Public Access)
```

The corresponding value for pattern property will be:

```
TNBXHotSpot1.Pattern="L - LIBRARY \(Public Access\)"
```

In this case we use \ character because parenthesis characters in regular expression has a different meaning. If you don't have regular expressions characters in the host screen field, you can assign the same text that you are viewing (without \ characters).
Associated with Pattern property, ActionFieldData property must be set. In this example the value of this second property will be:

TNBXHotSpot1.ActionFieldData="L@E"

See Also **ActionFieldData** property.

### 4.1.3.6.2.9 StartCol

Sets/gets the column where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

**Visual Basic Syntax**

*TNBXHotSpot.StartCol [= Long]*

**Delphi Syntax**

*TNBXHotSpot.StartCol [= LongInt];*

**Remarks**

Valid values are from 1 to *TNBX5250/TNBX3270.Cols* property.

See Also **EndRow, EndCol, StartRow** properties.

### 4.1.3.6.2.10 StartRow

Sets/gets the row where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

**Visual Basic Syntax**

*TNBXHotSpot.StartRow [= Long]*

**Delphi Syntax**

*TNBXHotSpot.StartRow [= LongInt];*

**Remarks**

Valid values are from 1 to *TNBX5250/TNBX3270.Rows* property.

See Also **EndRow, EndCol, StartCol** properties.
4.1.3.6.2.11 TextColor

Sets/gets the hotspot foreground color to be shown in a host screen.

**Visual Basic Syntax**

\[ TNBXHotSpot.TextColor = OLE\_COLOR \]

**Delphi Syntax**

\[ TNBXHotSpot.TextColor := OLE\_COLOR; \]

**Remarks**

This property is a long value that represent a hotspot foreground color.

See Also Background and ViewAs properties.

4.1.3.6.2.12 ViewAs

Sets or gets the display format of the hotspot. A hotspot can be displayed in four formats: plain text, link, button or none.

**Visual Basic Syntax**

\[ TNBXHotSpot.ViewAs = TXTnbHotSpotViewAs \]

**Delphi Syntax**

\[ TNBXHotSpot.ViewAs := TXTnbHotSpotViewAs; \]

It can take one of this constants values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsButton</td>
<td>Hotspot button style is displayed.</td>
</tr>
<tr>
<td>hsLink</td>
<td>Hotspot link style is displayed.</td>
</tr>
<tr>
<td>hsNone</td>
<td>No hotspot is displayed.</td>
</tr>
<tr>
<td>hsPlain</td>
<td>Hotspot plain text style is displayed.</td>
</tr>
</tbody>
</table>

4.1.3.6.3 Constants

4.1.3.6.3.1 TXTnbHotSpotViewAs

These values are used in the ViewAs property of hotspot control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsButton</td>
<td>Hotspot button style is displayed.</td>
</tr>
<tr>
<td>hsLink</td>
<td>Hotspot link style is displayed.</td>
</tr>
<tr>
<td>hsNone</td>
<td>No hotspot is displayed.</td>
</tr>
<tr>
<td>hsPlain</td>
<td>Hotspot plain text style is displayed.</td>
</tr>
</tbody>
</table>
4.1.3.7 TNBXHotSpots Component

4.1.3.7.1 TNBXHotSpots Component

Represent a collection of TNBXHotSpots controls. HotSpots allows you to work with mainframe applications using the mouse.

Properties

- BinData
- Count
- EndCol
- EndRow
- HotSpotName
- Items
- ProfilesControl
- StartCol
- StartRow
- SubKey
- ViewAs

Methods

- AboutBox
- Add
- AsVariant
- Delete
- LoadFromXMLFile
- SaveToXMLFile
- ShowEditor

4.1.3.7.2 Properties

4.1.3.7.2.1 BinData

Stores graphical information about hotspots on a form. Internal use only.

Visual Basic Syntax

TNBXHotSpots.BinData [= String]

Delphi Syntax

TNBXHotSpots.BinData [= String];

4.1.3.7.2.2 Count

Returns the total number of HotSpot controls contained in the HotSpots collection.

Visual Basic Syntax

[Integer =] TNBXHotSpots.Count

Delphi Syntax


[int: TNBXHotSpots.Count;

See Also HotSpot control.

4.1.3.7.2.3 EndCol

Sets/gets the column default value where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

[int: TNBXHotSpots.EndCol

**Delphi Syntax**

[int: TNBXHotSpots.EndCol;

**Remarks**

Return value is between 1 and TNBX5250/TNBX3270.Cols property.

See Also EndRow, StartCol, StartRow properties.

4.1.3.7.2.4 EndRow

Sets/gets the row default value where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

[int: TNBXHotSpots.EndRow

**Delphi Syntax**

[int: TNBXHotSpots.EndRow;

**Remarks**

Return value is between 1 and TNBX5250/TNBX3270.Rows property.

See Also EndCol, StartCol, StartRow properties.

4.1.3.7.2.5 HotSpotName

Sets/gets the description of the hotspot's default value. This is simply a descriptive string that will be associated with a hotspot at first.
**Visual Basic Syntax**

`TNBXHotSpot.HotSpotName [= String]`

**Delphi Syntax**

`TNBXHotSpot.HotSpotName [:= String];`

See Also [Description](#), [Id](#) properties.

---

### 4.1.3.7.2.6 Index

Returns or sets the number that uniquely identifies an object in a collection.

**Visual Basic Syntax**

`[Integer =] TNBXHotSpot.Index`

**Delphi Syntax**

`[Integer :=] TNBXHotSpot.Index;`

### 4.1.3.7.2.7 Items

Contains a collection of `TNBXHotSpot` objects.

**Visual Basic Syntax**

`TNBXHotSpots.Items(Index As Integer) [= TNBXHotSpot]`

**Delphi Syntax**

`TNBXHotSpots.Items[Index As Integer] [= TNBXHotSpot];`

**Remarks**

Use `Items` to get an specific hotspot from the collection. The `Index` parameter indicates the object's index in the collection, where 0 is the index of the first element and `(TNBXHotSpots.Count-1)` is the index of the last element. Use `Items` with the `Count` property to iterate through all the objects in the list.

See also `TNBHotSpot` control.
4.1.3.7.2.8 ProfilesControl

Sets the [TNBXProfiles] Control as the profile manager for this control.

**Visual Basic Syntax**

`TNBXHotSpots.ProfilesControl [= TNBXProfiles]`

**Delphi Syntax**

`TNBXHotSpots.ProfilesControl [:= TNBXProfiles];`

See Also [TNBXProfiles] control and [ShowEditor] method.

4.1.3.7.2.9 StartCol

Sets/gets the column default value where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

`[Integer =] TNBXHotSpots.StartCol`

**Delphi Syntax**

`[Integer :=] TNBXHotSpots.StartCol;`

**Remarks**

Return value is between 1 and [TNBX5250/TNBX3270. Cols] property.

See Also [EndRow], [EndCol], [StartRow] properties.

4.1.3.7.2.10 StartRow

Sets/gets the row default value where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

`[Integer =] TNBXHotSpots.StartRow`

**Delphi Syntax**

`[Integer :=] TNBXHotSpots.StartRow;`

**Remarks**
Return value is between 1 and `TNBX5250/TNBX3270.Rows` property.

See Also `EndCol`, `StartCol`, `EndRow` properties.

### 4.1.3.7.2.11 Subkey

Sets/gets the subkey under which will be stored the hotspot profile.

**Visual Basic Syntax**

```
TNBXHotSpots.SubKey [= String]
```

**Delphi Syntax**

```
TNBXHotSpots.SubKey := String;
```

**Remarks**

This property is valid only if a `ProfilesControl` property has been assigned to a `TNBXProfiles` control.

See also `ProfilesControl` property and `TNBXProfiles` Control.

### 4.1.3.7.2.12 ViewAs

Gets the display format default value of a hotspot. A hotspot can be displayed in four formats: plain text, link, button or none.

**Visual Basic Syntax**

```
[TXTnbHotSpotViewAs] TNBXHotSpot.ViewAs
```

**Delphi Syntax**

```
[TXTnbHotSpotViewAs] := TNBXHotSpot.ViewAs;
```

It can take one of this constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsButton</td>
<td>Hotspot button style is displayed.</td>
</tr>
<tr>
<td>hsLink</td>
<td>Hotspot link style is displayed.</td>
</tr>
<tr>
<td>hsNone</td>
<td>No hotspot is displayed.</td>
</tr>
<tr>
<td>hsPlain</td>
<td>Hotspot plain text style is displayed.</td>
</tr>
</tbody>
</table>

### 4.1.3.7.3 Methods

#### 4.1.3.7.3.1 AboutBox

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```
4.1.3.7.3.2 Add

Adds a new TnbXHotspot to the TNBXHotSpots collection.

**Visual Basic Syntax**

```vbnet
[TNBXHotSpot =] TNBXHotSpots.Add
```

**Delphi Syntax**

```delphi
[TNBXHotSpot :=] TNBXHotSpots.Add;
```

See Also `Delete` method.

4.1.3.7.3.3 AsVariant

Returns a HotSpots collection as a variant data type variable.

**Visual Basic Syntax**

```vbnet
[Variant =] TNBXHotSpots.AsVariant
```

**Delphi Syntax**

```delphi
[Variant :=] TNBXHotSpots.AsVariant;
```

4.1.3.7.3.4 Delete

Deletes a specified HotSpot of the TNBXHotSpots collection.

**Visual Basic Syntax**

```
TNBXHotSpots.Delete (Index As Integer)
```

**Delphi Syntax**

```
TNBXHotSpots.Delete (Index: Integer);
```

See Also `Add` method.

4.1.3.7.3.5 LoadFromXMLFile

Imports the XML code corresponding to the TNBXHotSpots object.

**Visual Basic Syntax**

```
```
LoadFromXMLFile (XMLFile As String)

**Delphi Syntax**

TNBXHotSpots.LoadFromXMLFile (XMLFile: string);

See also `SaveToXMLFile` method.

4.1.3.7.3.6  **SaveToXMLFile**

Exports the XML code corresponding to the `TNBXHotSpots` object.

**Visual Basic Syntax**

TNBXHotSpots.SaveToXMLFile (XMLFile As String)

**Delphi Syntax**

TNBXHotSpots.SaveToXMLFile (XMLFile: string);

See also `LoadFromXMLFile` method.

4.1.3.7.3.7  **ShowEditor**

Shows a modal dialog with the `TNBXHotSpot's` properties.

**Visual Basic Syntax**

TNBXHotSpots.ShowEditor

**Delphi Syntax**

TNBXHotSpots.ShowEditor;

See also `ProfilesControl` property.

4.1.4  **Helper Controls**

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack ActiveX Helper Controls.
4.1.4.1  TNBXSync Control

4.1.4.1.1  TNBXSync Control

This component gives a simplified interface for programming synchronously with TNB3270X and TNB5250X Controls (in a synchronous mode).

Properties

- EmptyScreenDelay
- TelnetControl
- TraceControl
- WaitTimeout

Methods

- AboutBox
- AsVariant
- Connect
- Disconnect
- IsConnected
- SendKeys
- Wait
- WaitFor
- WaitForConnect
- WaitForData
- WaitForDisconnect
- WaitForEndDoc
- WaitForNewScreen
- WaitForNewUnlock
- WaitForScreen
- WaitForStartDoc
- WaitForUnlock

4.1.4.1.2  Properties

4.1.4.1.2.1  EmptyScreenDelay

Sets/gets the additional wait time to bypass empty host screens. This time is measured in milliseconds.

Visual Basic Syntax

```
TNBXSync. EmptyScreenDelay [= Integer]
```

Delphi Syntax

```
TNBXSync. EmptyScreenDelay [= Integer];
```

Remarks

When you make a call to the `Wait` method, if an empty screen arrives from the host
and the system is in an unlocked state, an internal WaitForNewScreen is made with a timeout taken from this property value. It allows to bypass intermediate empty screens without additional programming effort. Applies only to the Wait method and other methods that call a Wait internally like Connect and SendKeys methods.

**Default** value is 0.

See also Connect, SendKeys and Wait methods.

### 4.1.4.1.2.2 TelnetControl

Sets the TNB3270X or TNB5250X Control as the telnet client control.

**Visual Basic Syntax**

```vbscript
TNBXSync.TelnetControl [= TNBnnnnX]
```

**Delphi Syntax**

```delphi
TNBXSync.TelnetControl [:= TNBnnnnX];
```

**Remarks**

You must set this property for TNBXSync control works properly.

### 4.1.4.1.2.3 TraceControl

Sets/gets the TNBXTrace Control as its trace agent.

**Visual Basic Syntax**

```vbscript
TNBXSync.TraceControl [= TNBXTrace]
```

**Delphi Syntax**

```delphi
TNBXSync.TraceControl [:= TNBXTrace];
```

**Remarks**

You must set this property to get trace information from TNBXSync control.

### 4.1.4.1.2.4 WaitTimeout

Sets/gets the default timeout value in milliseconds for all the wait methods.
Visual Basic Syntax

\[ \text{TNBXSync.WaitTimeout} = \text{Integer} \]

Delphi Syntax

\[ \text{TNBXSync.WaitTimeout} := \text{Integer} \];

Remarks

You can assign any value, there isn't a limit of maximum value.

Default value is 60000 milliseconds (equivalent to 1 minute).

See also \text{Wait, WaitFor, WaitForConnect, WaitForDisconnect, WaitForNewScreen, WaitForNewUnlock, WaitForScreen} and \text{WaitForUnlock} methods.

4.1.4.1.3 Methods

4.1.4.1.3.1 AboutBox

Shows Integration Pack about dialog box.

Visual Basic Syntax

\[ \text{TNBXSync.AboutBox} \]

Delphi Syntax

\[ \text{TNBXSync.AboutBox} ; \]

4.1.4.1.3.2 AsVariant

Returns the control as a variant data type variable.

Visual Basic Syntax

\[ \text{Variant} = \text{TNBXSync.AsVariant} \]

Delphi Syntax

\[ \text{Variant} := \text{TNBXSync.AsVariant} ; \]
4.1.4.1.3.3 Connect

The Connect method establishes a synchronous connection to the Telnet server.

**Visual Basic Syntax**

```vbscript
[Boolean] = TNBXSync.Connect([TimeOut])
```

**Delphi Syntax**

```delphi
[Boolean] := TNBXSync.Connect([TimeOut]);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the connection was successfully established and False if the Timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitForConnect` and `Wait` methods.

4.1.4.1.3.4 Disconnect

The Disconnect method terminates the connection to the Telnet server.

**Visual Basic Syntax**

```vbscript
[Boolean] = TNBXSync.Disconnect([TimeOut] As Integer)
```

**Delphi Syntax**

```delphi
[Boolean] := TNBXSync.Disconnect([TimeOut]: integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the disconnection was successfully established and False if the Timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitForDisconnect` and `Wait` methods.

4.1.4.1.3.5 IsConnected

Returns a boolean value indicating if a connection with the host is currently established.
4.1.4.1.3.6 SendKeys

The SendKeys method can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character ("@") and a mnemonic code that corresponds to the supported host functions.

**Visual Basic Syntax**

\[ \text{Boolean} = TNBXSync.IsConnected \]

**Delphi Syntax**

\[ \text{Boolean} := TNBXSync.IsConnected; \]

See also **Connect** and **Disconnect** methods.

### Remarks

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
</tbody>
</table>
This method complements the `SendKeys` method from `TNB3270X` controls and `TNB5250X`, adding the possibility to send several key sequences separated by aid keys and waits (@W). This is the main difference between both SendKeys methods. This mean that you can navigate through several screens with a single method call.

See also `TelnetControl` property of `TNBXSync` control and `SendKeys` method from `TNB3270X/TNB5250X` controls.

### 4.1.4.1.3.7 Wait

General wait mechanism. This method basically waits until the mainframe application turns in an unlocked state and is able to receive input data.

**Visual Basic Syntax**

```
[Boolean] = TNBXSync.Wait([TimeOut] As Integer)
```

**Delphi Syntax**

```
[Boolean] := TNBXSync.Wait([TimeOut]: integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the wait was successful and False if the timeout period has expired.

This method uses alternatively the `WaitForConnect`, `WaitForUnlock` and `WaitForNewScreen` methods, according to the actual mainframe state (disconnected, locked or unlocked).

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property, `WaitForConnect`, `WaitForUnlock` and `WaitForNewScreen` methods.
4.1.4.1.3.8 WaitFor

Wait for a screen containing the specified string.

**Visual Basic Syntax**

```vbnet
[Boolean] = TNBXSync.WaitFor(StrValue As String, [TimeOut] As Integer)
```

**Delphi Syntax**

```delphi
[Boolean] := TNBXSync.WaitFor(StrValue:string, [TimeOut]:integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the wait was successful and False if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property, `WaitForScreen`, `WaitForUnlock` and `WaitForNewScreen` methods.

4.1.4.1.3.9 WaitForConnect

This method waits until the telnet connection is successfully opened, or the timeout period expires.

**Visual Basic Syntax**

```vbnet
[Boolean] = TNBXSync.WaitForConnect([TimeOut] As Integer)
```

**Delphi Syntax**

```delphi
[Boolean] := TNBXSync.WaitForConnect([TimeOut]: integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the wait was successful and False if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `Wait` method.
4.1.4.1.3.10 WaitForData

Waits until data from the host printing process occurs. This method is used with Tnb3287X and Tnb3812X printer emulation controls and indicates that data has arrived.

Visual Basic Syntax

[Boolean] = TNBXSync.WaitForData ([TimeOut] As Integer)

Delphi Syntax

[Boolean] := TNBXSync.WaitForData ([TimeOut]: integer);

The TimeOut parameter is measured in milliseconds.

Remarks

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the wait was successful and False if the timeout period has expired.
If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

See also WaitTimeout property, WaitForScreen, WaitForUnlock and WaitForNewScreen methods.

4.1.4.1.3.11 WaitForDisconnect

This method waits until the telnet connection is successfully closed, or the timeout period expires.

Visual Basic Syntax

[Boolean] = TNBXSync.WaitForDisconnect ([TimeOut] As Integer)

Delphi Syntax

[Boolean] := TNBXSync.WaitForDisconnect ([TimeOut]: integer);

The TimeOut parameter is measured in milliseconds.

Remarks

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the wait was successful and False if the timeout period has expired.
If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.
See also `WaitTimeout` property and `Wait` method.

4.1.4.1.3.12 WaitForEndDoc

Waits until an end document event from host printed documents occurs. This method is used with `Tnb3287X` and `Tnb3812X` printer emulation controls and indicates the end of a document that was already printed.

**Visual Basic Syntax**

```
[Boolean] = TNBXSync.WaitForEndDoc ([TimeOut] As Integer)
```

**Delphi Syntax**

```
[Boolean] := TNBXSync.WaitForEndDoc ([TimeOut]: integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The `Timeout` parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the wait was successful and False if the timeout period has expired.

If you pass a timeout value in the `Timeout` parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `WaitForStartDoc` method.

4.1.4.1.3.13 WaitForNewScreen

This method waits until a new screen arrives within the specified period of time.

**Visual Basic Syntax**

```
[Boolean] = TNBXSync.WaitForNewScreen ([TimeOut] As Integer)
```

**Delphi Syntax**

```
[Boolean] := TNBXSync.WaitForNewScreen ([TimeOut]: integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

This method is similar to `WaitForScreen` method except that it always waits for a new screen arrival.

The `Timeout` parameter is optional, assuming the `WaitTimeout` property value by default. This method returns True if the wait was successful and False if the timeout period has expired.

If you pass a timeout value in the `Timeout` parameter of this method, the value of the property `WaitTimeout` will not have any effect.
See also **WaitTimeout** property and **WaitForScreen, WaitForData** methods.

### 4.1.4.1.3.14 WaitForNewUnlock

This method waits until a new system unlock arrives within the specified period of time.

**Visual Basic Syntax**

\[ [Boolean] = TNBXSync.WaitForNewUnlock ([Timeout] As Integer) \]

**Delphi Syntax**

\[ [Boolean] := TNBXSync.WaitForNewUnlock ([Timeout]:integer); \]

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

This method is similar to **WaitForUnlock** method except that it always waits for a new system unlock arrival.

The Timeout parameter is optional, assuming the **WaitTimeout** property value by default. This method returns True if the wait was successful and False if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property **WaitTimeout** will not have any effect.

See also **WaitTimeout** property and **WaitForUnlock** method.

### 4.1.4.1.3.15 WaitForScreen

This method waits for the first screen after a system lock.

**Visual Basic Syntax**

\[ [Boolean] = TNBXSync.WaitForScreen ([Timeout] As Integer) \]

**Delphi Syntax**

\[ [Boolean] := TNBXSync.WaitForScreen ([Timeout]:integer); \]

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

If the host system changes from an unlocked to a locked state (normally when we send an Aid key) the method waits until the first screen arrives. If WaitForScreen method is called after that event, it returns immediately.

The Timeout parameter is optional, assuming the **WaitTimeout** property value by default. The method returns True if the wait was successful and False if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the
property **WaitTimeout** will not have any effect.

See also **WaitTimeout** property and **WaitForNewScreen** method.

### 4.1.4.1.3.16 WaitForStartDoc

Waits until a start document event from the host, because of printed documents occurs. This method is used with **Tnb3287X** and **Tnb3812X** printer emulation controls and indicates the beginning of a document that was already printed.

**Visual Basic Syntax**

```vbnet
```

**Delphi Syntax**

```delphi
[Boolean] := TNBXSync.WaitForStartDoc ([Timeout]:integer);
```

The **TimeOut** parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the **WaitTimeout** property value by default. This method returns True if the wait was successful and False if the timeout period has expired.
If you pass a timeout value in the Timeout parameter of this method, the value of the property **WaitTimeout** will not have any effect.

See also **WaitTimeout** property and **WaitForEndDoc** method.

### 4.1.4.1.3.17 WaitForUnlock

This method waits until the host system turns to an unlocked state.

**Visual Basic Syntax**

```vbnet
[Boolean] = TNBXSync.WaitForUnlock ([Timeout] As Integer)
```

**Delphi Syntax**

```delphi
[Boolean] := TNBXSync.WaitForUnlock ([Timeout]:integer);
```

The **TimeOut** parameter is measured in milliseconds.

**Remarks**

If the host system is in an unlock state this method returns immediately. If the host system is in a lock state, the method waits until a system unlock event arrives (**OnSystemUnlock**).

The Timeout parameter is optional, assuming the **WaitTimeout** property value by
default. The method returns True if the wait was successful and False if the Timeout period has expired.
If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `WaitForNewUnlock` method.

4.1.4.2 TTNBXXmlTemplate Class

4.1.4.2.1 TTNBXXmlTemplate Class

The TTNbXmTemplate class implements the runtime side for working with XML Templates created with Development Lab. It implements the IXmlProducer interface and can be assigned to `Producer` property of `XmlBroker` to produce custom XML.

**Properties**

- **Id**

**Methods**

- `HostToXml`
- `XmlToHost`
- `CanProduce`

**Events**

- `OnCustomXml`

4.1.4.2.2 Properties

4.1.4.2.2.1 Id

Gets an Id for the active XML Template.

**VB.NET Syntax**

```vbnet
[String =] XmlTemplate.Id
```

**C# Syntax**

```csharp
[String =] XmlTemplate.Id;
```

**Remarks**

Returns a string with an identification corresponding to the active XML Template. This Id is equivalent to the filename of the XML template excluding the extension.
4.1.4.2.3 Methods

4.1.4.2.3.1 CanProduce

Returns true if the XmlTemplate can produce Xml.

**VB.NET Syntax**

```vbnet
[Boolean =] XmlTemplate.CanProduce
```

**C# Syntax**

```csharp
[Boolean =] XmlTemplate.CanProduce;
```

**Remarks**

When you call CanProduce method it tries to find an XML template that matches with the current Screen. In case it finds it, returns true telling that it can produce an XML content. In other case returns false.

4.1.4.2.3.2 HostToXml

Returns a XML representation of the current mainframe screen.

**VB.NET Syntax**

```vbnet
[String =] XmlTemplate.HostToXml
```

**C# Syntax**

```csharp
[String =] XmlTemplate.HostToXml;
```

**Remarks**

Returns an XML content based on the XML template that matches with the current screen.

4.1.4.2.3.3 XmlToHost

Interpretes the input XML to fill the unprotected fields.

**VB.NET Syntax**

```vbnet
XmlTemplate.HostToXml(Xml As String)
```

**C# Syntax**

```csharp
XmlTemplate.HostToXml(Xml: String);
```
Remarks

Interpretes the XML passed as parameter and fill unprotected fields, based on the active XML template for the current screen.

4.1.4.2.4 Events

4.1.4.2.4.1 OnCustomXml

Occurs during the processing of a custom XML area.

Remarks

When this event fires you have the chance to produce your custom XML content for that area.

4.1.4.3 TNBXXmlBroker Reference

4.1.4.3.1 TNBXXmlBroker Control

This component converts all data received from the TelnetControl component into XML code and when it receives XML code, sends it to the TelnetControl component.

Properties

- ScreenName
- TelnetControl
- XML

Methods

- LoadFromFile
- SaveToFile

See also XML Reference for TnbXXmlBroker.

4.1.4.3.2 Properties

4.1.4.3.2.1 IncludeStatus

Determines whether the Status tag and inner elements are included or not. Default value is true.

Visual Basic Syntax

TNBXXmlBroker.IncludeStatus [= Boolean]

Delphi Syntax

TNBXXmlBroker.IncludeStatus [= boolean];
**4.1.4.3.2.2 Producer**

Assigns an XML object capable to produce XML content.

**Visual Basic Syntax**

\[ TNBXXmlBroker.Producer = IXmlProducer \]

**Delphi Syntax**

\[ TNBXXmlBroker.Producer := IXmlProducer; \]

See also [TTNBXXmlTemplate](#) class.

**4.1.4.3.2.3 ScreenName**

Sets/gets the corresponding screen name.

**Visual Basic Syntax**

\[ TNBXXmlBroker.ScreenName = String \]

**Delphi Syntax**

\[ TNBXXmlBroker.ScreenName := String; \]

See also [XML Reference](#) for TnbXXmlBroker.

**4.1.4.3.2.4 TelnetControl**

Sets/gets the TNB3270X or TNB5250X ActiveX Control as the telnet client control.

**Visual Basic Syntax**

\[ TNBXXmlBroker.TelnetControl = TNBnnnnX \]

**Delphi Syntax**

\[ TNBXXmlBroker.TelnetControl := TNBnnnnX; \]

**4.1.4.3.2.5 XML**

When assigning an XML code, it generates a [TnbXFields](#) collection. Reading this property, generates and returns an XML code from unprotected [TnbXFields](#), cursor location and AID Key.

**Visual Basic Syntax**
4.1.4.3.3 Methods

4.1.4.3.3.1 LoadFromFile

Imports an XML file and then sets the XML property with it.

Visual Basic Syntax

TNBXXmlBroker.LoadFromFile (XMLFile As String)

Delphi Syntax

TNBXXmlBroker. LoadFromFile (XMLFile: string);

4.1.4.3.3.2 SaveToFile

Exports the XML code corresponding to the TNBXXmlBroker object with all the unprotected fields and the screen configuration data.

Visual Basic Syntax

TNBXXmlBroker. SaveToFile (XMLFile As String)

Delphi Syntax

TNBXXmlBroker. SaveToFile (XMLFile: string);

4.1.4.3.4 XML Reference

This XML code is generated automatically by the TnbXXmlBroker control. The following is the tags’ hierarchy it uses:

<TNBBRIDGE>
  <status>
    <direction... />
    <transaction... />
    <cursor_pos... />
    <session... />
    <state... />
    <xmlscreen... />
    <cursor_field... />
    <timestamp... />  
  <status />
  <screen>
    <rows... />
  </screen>
</TNBRIDGE>
4.1.4.3.4.1 XML Tags and Attributes

This is the root tag and includes all the other tags. These are the following:

- **Status**: indicates connection characteristics.
- **Screen**: indicates screen's characteristics and contains a collection of screen fields.

See also [XML Example](#).

Contains the following tags:

- **direction**: indicates INPUT or OUTPUT direction. When the XML is generated by the TnbXXmIBroker control it is set with OUTPUT value.
- **transaction**: indicates transaction identifier.
- **cursor_pos**: indicates cursor location.
- **session**: LU-LU (in session with VTAM application), noSession (not in session) or SSCPLU (in session with VTAM).
- **state**: indicates LOCKED or UNLOCKED state.
- **xmlscreen**: indicates the screen name. You must set this property using the [ScreenName](#) property.
- **cursor_field**: indicates the field's name where the cursor is located.
- **timestamp**: indicates the date the file was created.

See also [XML Example](#) and [ScreenName](#) property.

Contains the following tags:

- **rows**: indicates the number of rows of the screen.
- **cols**: indicates the number of columns of the screen.
- **type**: indicates terminal type (3270 or 5250).
- **model**: indicates terminal model.
- **fields**: contains a collection of fields.
See also XML Example.

4.1.4.3.4.2 XML Example

Here you can see an example of XML Code:

```xml
<TNBRIDGE>
  <status>
    <direction>OUTPUT</direction>
    <transaction>00A8DB70000002</transaction>
    <cursor_pos>830</cursor_pos>
    <session>LU-LU</session>
    <state>UNLOCKED</state>
    <xmlscreen name="Session" />
    <cursor_field>R11C30</cursor_field>
    <timestamp>2453360.14494451</timestamp>
  </status>
  <screen>
    <rows>24</rows>
    <cols>80</cols>
    <type>TN3270</type>
    <model>2E</model>
    <fields>
      <field name="R01C02">
        <name len="4">R1C2</name>
        <attr byte="E4" eab="08" display="y" prot="y" numeric="n"
          intense="n" mdt="n" />
        <value maxlen="8" len="8">KLGLGON1</value>
      </field>
      <field name="R01C11">
        <name len="5">R1C11</name>
        <attr byte="E4" eab="28" display="y" prot="y" numeric="n"
          intense="n" mdt="n" />
        <value maxlen="13" len="13">-------------</value>
      </field>
      ...
    </fields>
  </screen>
</TNBRIDGE>
```

4.1.4.4 TNBXXmlClient Reference

4.1.4.4.1 TNBXXmlClient Control

This control acts like a virtual TNB3270X/TNB5250X component. It generates the
TnbXFields necessary as input for the emulation control or any other user interface from
XML code as well generates the XML code from the modified TnbXFields, cursor location
and 'pressed' AID key. When used in conjunction with the TNBXXmlBroker component
allows build a bridge between the 3270/5250 Telnet components and the emulation
interface using XML streaming. This bridge could be implemented as a client/server
communication using TCP/IP protocols, Web Services, etc.
Properties

- **AidKey**
- **AidString**
- **ExcludeEmptyFields**
- **HotSpotsControl**
- **ScreenRow**
- **Text**
- **XML**

Methods

- **Connect**
- **Disconnect**
- **GetScreenRowEx**
- **GetScreenText**
- **IsConnected**
- **FindEditFields**
- **LoadFromXMLFile**
- **SaveToXMLFile**
- **SendAid**
- **SendKeys**
- **SetConnected**

Events

- **OnConnect**
- **OnConnectionFail**
- **OnConnectRemote**
- **OnDisconnect**
- **OnDisconnectRemote**
- **OnScreenChange**
- **OnSendAid**
- **OnSystemLock**
- **OnSystemUnlock**

4.1.4.4.2 Properties

4.1.4.4.2.1 AidKey

Sets the Attention Identifier Key and executes the **SendAid** method. Modified fields can be input fields (unprotected) or, sometimes protected fields, having the property Modified set to True.

**Visual Basic Syntax**

\[ TNBXmlClient.AIDKey [ = TNBXAid] \]

**Delphi Syntax**

\[ TNBXmlClient.AIDKey [ := TNBXAid]; \]
It can take any of TNBAidKey constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>akPF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>akPF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>akPF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>akPF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>akPF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>akPF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>akPF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>akPF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>akPF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>akPF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>akPF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>akPF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>akPF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>akPF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>akPF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>akPF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>akPF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>akPA1</td>
<td>PA1 key</td>
</tr>
<tr>
<td>akReset</td>
<td>Reset key</td>
</tr>
<tr>
<td>akPgUp</td>
<td>Page Up key</td>
</tr>
<tr>
<td>akPgDown</td>
<td>Page Down key</td>
</tr>
<tr>
<td>akPgLeft</td>
<td>Page Left key</td>
</tr>
<tr>
<td>akPgRight</td>
<td>Page Right key</td>
</tr>
<tr>
<td>akPrint</td>
<td>Print key</td>
</tr>
<tr>
<td>akHelp</td>
<td>Help key</td>
</tr>
</tbody>
</table>

See also [TNBXAid](#) constants.
4.1.4.4.2.2 AidString

Sets the corresponding Attention Identifier Key when a SendAid method is executed, and gets this value later if it is necessary.

**Visual Basic Syntax**

```
TNBXXmlClient.AIDString [= AidKey As String]
```

**Delphi Syntax**

```
TNBXXmlClient.AIDString [= AidKey:String];
```

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>PF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>PF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>PF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>PF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>PF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>PA1</td>
<td>PA1 key</td>
</tr>
</tbody>
</table>

See also **SendAid** method.
4.1.4.4.2.3 ExcludeEmptyFields
Sets/gets the possibility of excluding all empty fields.

**Visual Basic Syntax**

TNBXXmIClient.ExcludeEmptyFields [= Boolean]

**Delphi Syntax**

TNBXXmIClient.ExcludeEmptyFields [:= Boolean];

4.1.4.4.2.4 HotSpotsControl
Sets/gets the TNBXHotSpot component as the hotspots handler for this control.

**Visual Basic Syntax**

TNB XmIC Ient.TNBXHotSpots [= TNBXHotSpot]

**Delphi Syntax**

TNB XmIC Ient.TNBXHotSpots [:= TNBXHotSpot];

See Also TNBXHotSpot component.

4.1.4.4.2.5 ScreenRow
Gets the text contained into the specified row.

**Visual Basic Syntax**

[String =] TNBXXmIClient.ScreenRow (Index As Integer)

**Delphi Syntax**

[String :=] TNBXXmIClient.ScreenRow [Index:integer];

**Remarks**

Index parameter indicates the corresponding row number.

4.1.4.4.2.6 Text
Gets the text located in the column and row coordinates, with the length specified.

**Visual Basic Syntax**

[String =] TNBXXmIClient.Text (r As Integer, c As Integer, l As Integer)
Delphi Syntax

[String :=] TNBXXmlClient.Text [r, c, l: Integer];

Remarks

r and c indicate the row and column coordinates respectively and l the length of the string.

4.1.4.4.2.7 XML

When assigning an XML code, it generates a TnbXFields collection. Reading this property, generates and returns an XML code from unprotected TnbXFields, cursor location and AID Key.

Visual Basic Syntax

TNBXXmlClient.XML [= String]

Delphi Syntax

TNBXXmlClient.XML [:= String];

4.1.4.4.3 Methods

4.1.4.4.3.1 Connect

The Connect method initializes the connection sequence process. Actually it doesn't produce by itself a connection to a resource, but fires the OnConnectRemote event in order to give the change to the client program to establish the connection to the resource which will be the XML producer (a local or remote file, a local or remote TnbXXMLBroker, a Web Service, etc).

Visual Basic Syntax

TNBXXmlClient.Connect

Delphi Syntax

TNBXXmlClient.Connect;

See also Disconnect method, OnConnect, OnConnectRemote and OnConnectionFail events.

4.1.4.4.3.2 Disconnect

The Disconnect method ends the connection sequence process. It actually doesn't produce by itself a disconnection to a resource, but fires the OnDisconnectRemote event in order to give the change to the client program to end the connection to the resource.
Visual Basic Syntax

TNBXXmlClient.Disconnect

Delphi Syntax

TNBXXmlClient.Disconnect;

See also Connect method, OnDisconnectRemote and OnDisconnect event.

4.1.4.4.3.3 GetScreenRowEx

GetScreenRowEx can be used to retrieve the text buffer of one row of the current screen. Additionally, you can retrieve the standard or extended attribute corresponding to each field.

Visual Basic Syntax

[String ] = TNBXXmlClient.GetScreenRowEx (Row As Integer, [Attr As Boolean = True], [Eab As Boolean = True])

Delphi Syntax

[String ] := TNBXXmlClient.GetScreenRowEx (Row:integer, [Attr:boolean = True], [Eab:boolean = True]);

Remarks

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specify. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned string, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

See also GetScreenText method.

4.1.4.4.3.4 GetScreenText

GetScreenText returns the text buffer of a portion of the current screen. You can choose to retrieve the attributes or the extended attributes of each field joined to the character data.

Visual Basic Syntax

[String ] = TNBXXmlClient.GetScreenText (StartPos As Integer, EndPos As Integer, [Attr

© 2015, Cybele Software, Inc.
Delphi Syntax

[String :=] TNBXXmIClient.GetScreenText (StartPos:integer, EndPos:integer,
[Attr:boolean = True], [Eab:boolean = True]);

Remarks

StartPos and EndPos specify the start position and the end position of the text buffer that will be returned.
The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.
The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in an 80x24 display, each row returned will occupy 160 characters.

See also GetScreenRowEx method.

4.1.4.3.5 IsConnected

Returns a boolean value indicating if the emulation is on process.

Visual Basic Syntax

[Boolean =] TNBXXmIClient.IsConnected

Delphi Syntax

[Boolean :=] TNBXXmIClient.IsConnected;

See also Connect and Disconnect methods.

4.1.4.3.6 FindEditField

Gets the specified unprotected field, searching it by its name.

Visual Basic Syntax

[TTnbField =] TNBXXmIClient.FindEditField (caption As String)

Delphi Syntax

[TTnbField :=] TNBXXmIClient.FindEditField (caption:String);
Remarks

Caption parameter specifies the name of the label associated with the field.

4.1.4.4.3.7 LoadFromXMLFile

Imports an XML file and then sets the XML property with it.

**Visual Basic Syntax**

\[TNBXXmlClient.LoadFromXMLFile (XMLFile As String)\]

**Delphi Syntax**

\[TNBXXmlClient.LoadFromXMLFile (XMLFile: String);\]

4.1.4.4.3.8 SaveToXMLFile

Exports the XML code corresponding to the TNBXXmlClient object with all the unprotected fields and the screen configuration data.

**Visual Basic Syntax**

\[TNBXXmlClient.SaveToXMLFile (XMLFile As String)\]

**Delphi Syntax**

\[TNBXXmlClient.SaveToXMLFile (XMLFile: string);\]

4.1.4.4.3.9 SendAid

Sets an Attention Identifier Key to the AidString property and fires the OnSendAid event.

**Visual Basic Syntax**

\[TNBXXmlClient.SendAid (AidKey As string)\]

**Delphi Syntax**

\[TNBXXmlClient.SendAid (AidKey:string);\]

This method takes a string with the name of the AID key as a parameter. You can specify only one key for each call to the method.
The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>PF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>PF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>PF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>PF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>PF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>PA1</td>
<td>PA1 key</td>
</tr>
</tbody>
</table>

### 4.1.4.3.10 SendKeys

SendKeys can be used to send key sequences of characters (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host functions.

#### Visual Basic Syntax

```vbnet
TNBXXmlClient.SendKeys (Keys As String)
```

#### Delphi Syntax

```delphi
TNBXXmlClient.SendKeys (Keys:string);
```
The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the Keys parameter:

"logon applid(cics)@E"

**Remarks**

The execution of the SendKeys is stopped when an AID Key is sent and when this happen, the AidKey property is set with this AidKey value.
4.1.4.4.4.3.11 SetConnected

Sets the connection state according to the parameter received.

**Visual Basic Syntax**

```
TNBXXmlClient.SetConnected [= Boolean]
```

**Delphi Syntax**

```
TNBXXmlClient.SetConnected [= Boolean];
```

4.1.4.4.4 Events

4.1.4.4.4.1 OnConnect

Occurs when the program called the `SetConnected` method with true, indicating that the connection has been successfully established.

See also `Connect` and `SetConnected` methods, `OnConnectRemote`, `OnConnectionFail`, `OnDisconnect` and `OnDisconnectRemote` events.

4.1.4.4.4.2 OnConnectionFail

Occurs when the program called the `SetConnected` method with false, indicating that the connection couldn't be established.

See also `Connect` method, `OnConnect` and `OnDisconnect` events.

4.1.4.4.4.3 OnConnectRemote

Occurs as a consequence of the `Connect` method execution. You must set the `SetConnected` property to `True` or `False`, according to the result of the connection to the final resource.

See also `Connect` and `SetConnected` methods, `OnConnect`, `OnConnectionFail`, `OnDisconnect` and `OnDisconnectRemote` events.

4.1.4.4.4.4 OnDisconnect

Occurs when the `SetConnected` property is set to `False`.

See also `Disconnect` and `SetConnected` methods, `OnConnect`, `OnConnectRemote`, `OnConnectionFail`, `OnDisconnect` and `OnDisconnectRemote` events.

4.1.4.4.4.5 OnDisconnectRemote

Occurs as a consequence of the `Disconnect` method execution. You must set the `SetConnected` property to `True` or `False`, according to the result of the disconnection to the final resource.

Copyright © 2002-2008, Cybele Software Inc. All rights reserved.
See also Disconnect and SetConnected methods, OnConnect, OnDisconnect, OnConnectRemote, OnDisconnectRemote events.

### 4.1.4.4.4.6 OnScreenChange

Occurs when an valid XML is assigned, either thru XML property or LoadFromXmlFile method.

See also XML property, OnSystemLock and OnSystemUnlock events.

### 4.1.4.4.4.7 OnSendAid

Occurs before an AID key is about to be sent.

**Remarks**

This event is fired when a SendAid or SendKeys methods or the AidKey property are used. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator component.

See also SendAid, SendKeys methods and AidKey property.

### 4.1.4.4.4.8 OnSystemLock

Occurs when the XML stream changes to a locked state.

**Remarks**

During this state, sending data isn't possible until the OnSystemUnlock event is fired.

See also OnSystemUnlock event.

### 4.1.4.4.4.9 OnSystemUnlock

Occurs when the XML stream changes to an unlocked state.

**Remarks**

During this state, sending data is possible until the OnSystemLock event occurs.

See also OnSystemLock event.

### 4.1.4.5 TNBXXmlVirtual Reference

#### 4.1.4.5.1 TNBXXmlVirtual Control

This component allows you to create a simulated host application by combining XML-screen files and code to drive the screen navigation. XML-screen files can be taken using Development Lab.

**Properties**
- AidKey
- BaseDirectory
- CurrentScreen
- ExcludeEmptyFields
- FieldSplitting
- StartFilename
- Text

**Methods**

- Connect
- Disconnect
- GetScreenText
- IsConnected
- LoadScreen
- Press
- Type
- SetConnected

**Events**

- OnConnect
- OnConnectionFail
- OnDisconnect
- OnScreenChange
- OnSendAid
- OnSystemLock
- OnSystemUnlock
- OnNewScreen

4.1.4.5.2 Properties

4.1.4.5.2.1 AidKey

Sets the Attention Identifier Key and executes the SendAid method. Modified fields can be input fields (unprotected) or, sometimes protected fields, having the property Modified set to True.

**Visual Basic Syntax**

```
TNBXmlVirtual.AIDKey [= TNBAid]
```

**Delphi Syntax**

```
TNBXmlVirtual.AIDKey [ := TNBAid];
```

It can take any of AidKey constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
</tbody>
</table>
akEnter | ENTER key  
akClear | CLEAR key  
akATTN | Attention key  
akSysReq | System Request key  
akPF1 | PF01 key  
akPF2 | PF02 key  
akPF3 | PF03 key  
akPF4 | PF04 key  
akPF5 | PF05 key  
akPF6 | PF06 key  
akPF7 | PF07 key  
akPF8 | PF08 key  
akPF9 | PF09 key  
akPF10 | PF10 key  
akPF11 | PF11 key  
akPF12 | PF12 key  
akPF13 | PF13 key  
akPF14 | PF14 key  
akPF15 | PF15 key  
akPF16 | PF16 key  
akPF17 | PF17 key  
akPF18 | PF18 key  
akPF19 | PF19 key  
akPF20 | PF20 key  
akPF21 | PF21 key  
akPF22 | PF22 key  
akPF23 | PF23 key  
akPF24 | PF24 key  
akPA1 | PA1 key  
akReset | Reset key  
akPgUp | Page Up key  
akPgDown | Page Down key  
akPgLeft | Page Left key  
akPgRight | Page Right key  
akPrint | Print key  
akHelp | Help key

See also Aid constants.

4.1.4.5.2.2 BaseDirectory

Gets/sets the base directory from where Screen XML files will be loaded.

**Visual Basic Syntax**

```vbnet
TNBXmIVirtual.BaseDirectory [= BaseDirectory As String]
```

**Delphi Syntax**
TNBXmlVirtual.BaseDirectory [:: BaseDirectory:String];

**Remarks**

BaseDirectory is used to automatically load screen files when you press PgDown/PgUp keys.

See also **OnNewScreen** event and **StartFilename** property.

4.1.4.5.2.3 CurrentScreen

Gets the name of current XML filename used to render the current screen.

**Visual Basic Syntax**

TNBXmlVirtual.CurrentScreen [:: CurrentScreen As String]

**Delphi Syntax**

TNBXmlVirtual.CurrentScreen [:: CurrentScreen:String];

4.1.4.5.2.4 ExcludeEmptyFields

Sets/gets the possibility of excluding all empty fields.

**Visual Basic Syntax**

TNBXmlVirtual.ExcludeEmptyFields [:: ExcludeEmptyFields As Boolean]

**Delphi Syntax**

TNBXmlVirtual.ExcludeEmptyFields [:: ExcludeEmptyFields:Boolean];

4.1.4.5.2.5 FieldSplitting

Determines the format of Fields in HostFields and EditFields collections.

**Visual Basic Syntax**

TNBXmlVirtual.FieldSplitting [:: FieldSplitting As Boolean]

**Delphi Syntax**

TNBXmlVirtual.FieldSplitting [:: FieldSplitting:Boolean];

**Remarks**

FieldSplitting property controls how fields in HostFields and EditFields collections are constructed. When active, fields that go beyond the maximum of display column are
splitted in two or more fields.

Default is true.

4.1.4.5.2.6  StartFilename

Gets/Sets the start XML File.

**Visual Basic Syntax**

\[
TNBXmIVirtual.StartFileName \[= \text{StartFileName As String}\]
\]

**Delphi Syntax**

\[
TNBXmIVirtual.StartFileName [:= \text{StartFileName:String}];
\]

**Remarks**

When you call the connect method the StartFilename will be the one that will be used to provide the content for the first screen. BaseDirectory will be set to the directory where StartFilename belongs.

See also OnNewScreen event and BaseDirectory property.

4.1.4.5.2.7  Text

Returns the entire screen.

**VB Syntax**

\[
\text{String} = TNBXmIVirtual.Text
\]

**Delphi Syntax**

\[
\text{String} = TNBXmIVirtual.Text;
\]

4.1.4.5.3  Methods

4.1.4.5.3.1  Connect

The Connect method initializes the connection sequence process. It loads the StartFilename XML File.

**Visual Basic Syntax**

\[
TNBXmIVirtual.Connect
\]

**Delphi Syntax**

\[
TNBXmIVirtual.Connect;
\]
See also **Disconnect** method, **OnConnect** and **OnConnectionFail** events.

### 4.1.4.5.3.2 Disconnect

The Disconnect method ends the connection sequence process.

**Visual Basic Syntax**

```vbnet
TNBXmlVirtual.Disconnect
```

**Delphi Syntax**

```delphi
TNBXmlVirtual.Disconnect;
```

See also **Connect** method and **OnDisconnect** event.

### 4.1.4.5.3.3 IsConnected

Returns a boolean value indicating if the emulation is on process.

**VB Syntax**

```vbnet
[Boolean =] TNBXmlVirtual.IsConnected
```

**Delphi Syntax**

```delphi
[boolean :=] TNBXmlVirtual.IsConnected;
```

See also **Connect** and **Disconnect** methods.

### 4.1.4.5.3.4 IsLocked

Returns a boolean value indicating if the Host is in a locked state.

**VB Syntax**

```vbnet
[Boolean =] TNBXmlVirtual.IsLocked
```

**Delphi Syntax**

```delphi
[boolean :=] TNBXmlVirtual.IsLocked;
```

### 4.1.4.5.3.5 IsScreenEmpty

Returns a boolean value indicating in the screen has all the fields with no content or in filled with space characters.
VB Syntax

[Boolean =] TNBXmlVirtual.IsScreenEmpty

Delphi Syntax

[boolean :=] TNBXmlVirtual.IsScreenEmpty;

4.1.4.5.3.6 GetText

GetText returns the text buffer of a portion of the current screen. You can receive the attribute or extended attribute of each field together to the character data.

VB Syntax

[String =] TNBXmlVirtual.GetText (StartPos As Integer, [EndPos As Integer], [Attr As Boolean = False], [Eab As Boolean = False])

[String =] TNBXmlVirtual.GetText (Row As Integer, Col As Integer, Len as Integer,[Attr As Boolean = False], [Eab As Boolean = False])

Delphi Syntax

[string :=] TNBXmlVirtual.GetText (int StartPos, [int StartPos], [bool Attr = False],
[bool Eab = False]);

[string :=] TNBXmlVirtual.GetText (int Row, in Col, int Len, [bool Attr = False], [bool Eab = False]);

Remarks

Start position and the end position of the text buffer to retrieve can be set using StartPos and EndPos parameters or Row,Col and Len parameters.

The Attr parameter will indicate to GetText that includes the field attribute information. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter will indicate to GetText that includes the extended attribute information. The Eab is associated with individual characters. Each character in the buffer (except for fields attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format.

4.1.4.5.3.7 LoadScreen

Loads an XML file to provide with new screen content.

VB Syntax
4.1.4.5.3.8 Press

Sets an Attention Identifier Key and sends the modified fields to the Mainframe. Modified fields can be input fields (unprotected) or sometimes protected fields, having the property Modified set to True.

**VB Syntax**

```vb
[Boolean =] TNBXmIVirtual.Press (Value As string)
```

**Delphi Syntax**

```delphi
[boolean :=] TNBXmIVirtual.Press (xmlFileName:string);
```

This method works like the AidKey property, but it takes a string with the name of the AID key as a parameter. You can specify only one key for each call to the method.

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
</tbody>
</table>
4.1.4.5.3.9 SetConnected
Sets the connection state according to the parameter received.

**VB Syntax**

TNBXmlVirtual.SetConnected (Value As Boolean)

**Delphi Syntax**

TNBXmlVirtual.SetConnected (value:boolean);

4.1.4.5.3.10 SetText
SetText allows to put text information into edit buffer.

**VB Syntax**

TNBXmlVirtual.SetText (StartPos As Integer, Text As String)

TNBXmlVirtual.SetText (Row As Integer; Col As Integer; Text As String)

**Delphi Syntax**

TNBXmlVirtual.SetText (StartPos:integer; Text:string)

TNBXmlVirtual.SetText (Row:integer; Col:integer; Text:string)

**Remarks**

SetText allows to put text information into edit buffer, starting according to position StartPos or Row and Col parameters.

4.1.4.5.3.11 Type
Type can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host functions.

**VB Syntax**

TNBXmlVirtual.Type (DataString As String)
Delphi Syntax

TNBXMLVirtual.Type (DataString:string);

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the Keys parameter:

"logon applid(cics)@E"

Remarks

The execution of the SendKeys is stopped when an AID Key is sent to the Host.
4.1.4.5.4 Events

4.1.4.5.4.1 OnConnect

Occurs when the program called the SetConnected method with true, indicating that the connection has been successfully established.

See also Connect and SetConnected methods, OnConnectionFail and OnDisconnect events.

4.1.4.5.4.2 OnConnectionFail

Occurs when the program called the SetConnected method with false, indicating that the connection couldn't be established.

See also Connect method, OnConnect and OnDisconnect events.

4.1.4.5.4.3 OnDisconnect

Occurs when the SetConnected property is set to False.

See also Disconnect and SetConnected methods, OnConnect and OnDisconnect event.

4.1.4.5.4.4 OnNewScreen

Occurs when a new XML file should be loaded.

Remarks

When this event is fired, you can analize AidKey to determine the next screen you should load using LoadScreen method.

See also LoadScreen method.

4.1.4.5.4.5 OnScreenChange

Occurs when an valid XML is assigned, either thru XML is property or LoadFromXmlFile method.

See also XML property, OnSystemLock and OnSystemUnlock events.

4.1.4.5.4.6 OnSendAid

Occurs before an AID key is about to be sent.

Remarks

This event is fired when a SendAid or SendKeys methods or the AidKey property are used. Occurs before the data is sent, allowing to take actions like filling edit fields or
saving data typed in the emulator component.

See also SendAid, SendKeys methods and AidKey property.

4.1.4.5.4.7 OnSystemLock

Occurs when the XML stream changes to a locked state.

Remarks

During this state, sending data isn't possible until the OnSystemUnlock event is fired.

See also OnSystemUnlock event.

4.1.4.5.4.8 OnSystemUnlock

Occurs when the XML stream changes to an unlocked state.

Remarks

During this state, sending data is possible until the OnSystemLock event occurs.

See also OnSystemLock event.

4.1.5 Trace Services

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack ActiveX Trace Services.

4.1.5.1 Setting up Trace Services

To trace a TN Bridge application you need to setup the Trace Agent built into the TNBXTrace Control.

The following steps shows you how to setup the Trace Services to work with your application:

- Drop a TNBXTrace control into a form.
- Set the Telnet Control property of the trace control to the Telnet control:
  
  \[
  \text{TNBXTrace1.TelnetControl = Telnet}
  \]
- Set the TCP/IP listening port to the Port property (at design time or at run time):
  
  \[
  \text{TNBXTrace1.Port = 1024}
  \]
- Set the Active property to true. When doing this at design-time, the activation takes place at run-time:
  
  \[
  \text{TNBXTrace1.Active = True}
  \]
- Run your application.
Each time the telnet control connects to a mainframe, a trace file is generated with a connection identifier as its name and extension the ".hst". This file will be stored into the folder specified in TNBXTrace Directory property or in windows temporary files folder. By default, this files will be removed when the trace becomes inactive, but you can change this behavior setting the PurgeTraceFiles property to False.

For on-line monitoring:

- Open the TN Bridge Trace Viewer and setup a connection pointing to the machine IP address where your TN Bridge application is running. Also set the TCP Port to the listening port specified in your TNBXTrace control. This is the Connections Settings dialog box where you can complete the information:

```
Connections Settings

Name: Mi queido HOST
Host: 127.0.0.1
Port: 1024

Add Modify Delete Close
```

- Click the connect button. A dialog box with the available connections will be shown. Choose one from the list and click the Connect button.
For off-line monitoring:

- Open the TN Bridge Trace Viewer and choose the file menu and open menu item. Select the trace file previously generated by your TN Bridge application. (Remember that this file will be available only if you set the TNBXTrace's PurgeTraceFiles property to False. Also, it is strongly recommended to set the Directory property to an existing a known folder).

4.1.5.2 TN Bridge Trace Server

4.1.5.2.1 TNBXTrace Control

This control implements a Trace Agent. It allows real-time monitoring of Telnet events, analyses mainframe's screen information and TNBXSync methods calls.

**Properties**

- Active
- Directory
- HidePasswordFields
- Port
- PurgeTraceFiles
- TelnetControl
- Visible

**Methods**

- AboutBox
- AsVariant
- HideTrace
- ShowTrace
- Trace
- TraceEx
4.1.5.2.2 Properties

4.1.5.2.2.1 Active

Activates or deactivates Trace Agent functionality.

**Visual Basic Syntax**

TNBXTrace.Active [ = Boolean]

**Delphi Syntax**

TNBXTrace.Active; [: = Boolean]

**Remarks**

If you assign the value of this property at design time, it acts only as the initial value for run-time.

Default value is **False**.

4.1.5.2.2.2 Directory

Sets/gets the folder directory name where the trace files will be saved.

**Visual Basic Syntax**

TNBXTrace.Directory [ = String]

**Delphi Syntax**

TNBXTrace.Directory; [: = String]

**Remarks**

If you don't specify a directory name, the trace files will be saved into the windows temporary file directory.
4.1.5.2.2.3  HidePasswordFields

Enables or disables the password masking mode for hiding data of password fields.

**Visual Basic Syntax**

```
TNBXTrace.HidePasswordFields [= Boolean]
```

**Delphi Syntax**

```
TNBXTrace.HidePasswordFields; [:= boolean]
```

**Remarks**

When this property is set to **True**, password fields are masked with asterisks.

Default value is **True**.

4.1.5.2.2.4  Port

Sets/gets the listening TCP port number for the current application process. The TNBXTrace control works at process level, generating independent traces on a connection basis.

**Visual Basic Syntax**

```
TNBXTrace.Port [= Integer]
```

**Delphi Syntax**

```
TNBXTrace.Port; [:= Integer]
```

**Remarks**

The default port value is 1024.

4.1.5.2.2.5  PurgeTraceFiles

Allows to purge trace files when the server becomes inactive.

**Visual Basic Syntax**
**TNBXTrace.PurgeTraceFiles** [= Boolean]

**Delphi Syntax**

`TNBXTrace.PurgeTraceFiles; [: = Boolean]

**Remarks**

By default, trace information is kept into temporary files. When the **Active** property is set to **False** or the process is finished, these files are deleted.

Default value is **True**.

**4.1.5.2.2.6 TelnetControl**

Sets the TNB3270X or TNB5250X Control as the telnet client control.

**Visual Basic Syntax**

`TNBXTrace.TelnetControl [= TNBnnnn]`

**Delphi Syntax**

`TNBXTrace.TelnetControl; [: = TNBnnnn]`

**Remarks**

You must set this property to allow TNBXTrace component to work properly.

**4.1.5.2.2.7 Visible**

Activates or deactivates Trace Viewer window. It can be used at the same time you are developing an Integration Pack application, helping about host events that happen when application code is running. Trace Viewer window helps to follow those host events as you modify application's code.
**Visual Basic Syntax**

`TNBXTrace.Visible [= Boolean]`

**Delphi Syntax**

`TNBXTrace.Visible; [= Boolean]`

**Remarks**

If you set the value of this property at design time to True, Trace Viewer window is shown before you run Integration Pack application, and will be prepared to show hosts events at application run-time.

Default value is **False**.

---

4.1.5.2.3 Methods

4.1.5.2.3.1 AboutBox

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

`TNBXTrace.AboutBox`

**Delphi Syntax**

`TNBXTrace.AboutBox;`
4.1.5.2.3.2 AsVariant

Returns the control as a variant data type variable.

**Visual Basic Syntax**

```vbnet
Variant = TNBXTrace.AsVariant
```

**Delphi Syntax**

```delphi
Variant := TNBXTrace.AsVariant;
```

4.1.5.2.3.3 HideTrace

Hides Trace Viewer window. It can be used at the same time you are developing an Integration Pack application, helping you with host events that happen when application code is running.

**Visual Basic Syntax**

```vbnet
TNBXTrace.HideTrace
```

**Delphi Syntax**

```delphi
TNBXTrace.HideTrace;
```

**Remarks**

This method is used in developing Integration Pack’s application process.

See also **ShowTrace** method and **Visible** property.

4.1.5.2.3.4 ShowTrace

Shows Trace Viewer window. It can be used at the same time you are developing an Integration Pack application, helping about host events that happen when application code is running.

**Visual Basic Syntax**

```vbnet

```
**TNBXTrace.ShowTrace**

**Delphi Syntax**

`TNBXTrace.ShowTrace;`

**Remarks**

This method is used in developing Integration Pack's application process.

See also **HideTrace** method and **Visible** property.

---

4.1.5.2.3.5 Trace

Allows to insert custom string data into the trace file.

**Visual Basic Syntax**

`TNBXTrace.Trace (Message As String)`

**Delphi Syntax**

`TNBXTrace.Trace (Message:string);`

**Remarks**

This method can be used for debugging purposes by making an entry with String content into the trace file.

---

4.1.5.2.3.6 TraceEx

Allows to insert custom string data into the trace file.

**Visual Basic Syntax**

`TNBXTrace.TraceEx (Kind as Integer, Msg as String)`

**Delphi Syntax**

`TNBXTrace.TraceEx (Kind:integer; Msg:string);`
Remarks

This method can be used for debugging purposes by making an entry with String content into the trace file. Also has a kind parameter to identify the message kind.

The following table lists different values for kind parameter and its meaning.

<table>
<thead>
<tr>
<th>Long Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Error</td>
</tr>
<tr>
<td>1</td>
<td>Info</td>
</tr>
<tr>
<td>2</td>
<td>StartPage</td>
</tr>
<tr>
<td>3</td>
<td>EndPage</td>
</tr>
<tr>
<td>4</td>
<td>StartMethod</td>
</tr>
<tr>
<td>5</td>
<td>EndMethod</td>
</tr>
<tr>
<td>6</td>
<td>SetProperty</td>
</tr>
<tr>
<td>7</td>
<td>SetVariable</td>
</tr>
<tr>
<td>8</td>
<td>Event</td>
</tr>
<tr>
<td>9</td>
<td>FieldsReceived</td>
</tr>
<tr>
<td>10</td>
<td>SendingFields</td>
</tr>
<tr>
<td>11</td>
<td>Connected</td>
</tr>
<tr>
<td>12</td>
<td>Disconnected</td>
</tr>
<tr>
<td>13</td>
<td>SendAid</td>
</tr>
</tbody>
</table>

4.2 Delphi Components

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack Delphi components.

- Syntax Conventions
- Mainframe Access Components
- Terminal Emulator Components
- Helper Controls
- Trace Services

4.2.1 Syntax conventions

The following text formats are used throughout the Components Reference:

For each property and method:

*Italic text*  Denotes an object, in this case a TN Bridge component.
*Bold text*    Denotes a property or a method names.
*[: = value]*  Denotes values to be assigned.
4.2.2 Mainframe Access Components

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack Delphi Mainframe Access Components.

4.2.2.1 TTnb5250/TTnb3270 Reference

4.2.2.1.1 TTnb5250/TTnb3270 Components

Both components share a common programming interface, except for a few properties that are specific to one component.

Properties

- **AIDKey**
- **CodePage**
- **Cols**
- **ConnectionName**
- **ConversionTable**
- **Debug**
- **DebugDir**
- **DeviceName**
- **EditFields**
- **ExcludeEmptyFields**
- **Extended**
- **Host**
- **HostFields**
- **IsExtended**
- **KeepAlive**
- **LastErrMsg**
- **Password**
- **Port**
- **Rows**
- **ScreenRow**
- **SessionState**
- **SubKey**
- **TerminalType**
- **TnbProfiles**
- **TnbTrace**
- **UserID**
- **XLockDelay**

Methods

- **About**
- **ClassNameIs**
- **CloseEditor**
- **Connect**
- **Disconnect**
• GetScreenRowEx
• GetScreenText
• IsConnected
• LoadFromXMLFile
• SaveToXMLFile
• SendAid
• SendKeys
• ShowEditor

Events

• OnConnect
• OnConnectionFail
• OnDisconnect
• OnScreenChange
• OnSendAid
• OnSessionState
• OnSystemLock
• OnSystemUnlock

4.2.2.1.2 Properties

4.2.2.1.2.1 AidKey

Sets the Attention Identifier Key and sends the modified fields to the Mainframe. Modified fields can be input fields (unprotected) or, sometimes protected fields, having the property Modified set to True.

Visual Basic Syntax

TNBnnnn.AIDKey [= TNBAid]

Delphi Syntax

TNBnnnn.AIDKey [:= TNBAid];

It can take any of TNBAidKey constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
</tbody>
</table>
See also \texttt{TNBXAid} constants.

4.2.2.1.2.2 CodePage

Specifies the internal ASCII-EBCDIC conversion table.

\textbf{Visual Basic Syntax}

\texttt{TNBnnnn.CodePage [= Integer]}

\textbf{Delphi Syntax}

\texttt{TNBnnnn.CodePage ::= Integer;}

\textbf{Remarks}

It can take any of the Code Pages constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia = 037</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium = 037</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada = 037</td>
<td>Canada</td>
</tr>
</tbody>
</table>
4.2.1.2.3 Cols

Return the total number of columns of the current terminal type.

**Visual Basic Syntax**

\[ \text{Integer} = \text{TNBnnnn}.Cols \]

**Delphi Syntax**

\[ \text{Integer} := \text{TNBnnnn}.Cols; \]

**Remarks**

The Cols property may return 80 or 132 values depending on the terminal type chosen.

See also **Rows** and **TerminalType** properties.
4.2.2.1.2.4 ConnectionName

Sets/gets the connection name under which will be stored the connection.

**Visual Basic Syntax**

TNBnnnn.ConnectionName [= String]

**Delphi Syntax**

TNBnnnn.ConnectionName [= String];

The default values are "TNB3270E" for TNB3270X Control and "TNB5250" for TNB5250X Control.

**Remarks**

This property is valid only if a ProfilesControl property has been assigned to a TNBProfiles Control.

See also ProfilesControl property and TNBProfiles Component.

4.2.2.1.2.5 ConversionTable

Specifies an external ASCII-EBCDIC conversion table to be loaded from a file. If no one is specified, an internal conversion table is used, according to CodePage property setting.

**Visual Basic Syntax**

TNBnnnn.ConversionTable [= String]

**Delphi Syntax**

TNBnnnn.ConversionTable [= String];

**Remarks**

The file specified must exist and it might contain a valid conversion table format.

The format of the conversion table is:
4.2.2.1.2.6 Debug

Allows the generation of trace information for debugging purposes. The debugging can help you to track down both, errors and logic information.

**Visual Basic Syntax**

```
TNBnnnn.Debug [= Boolean]
```

**Delphi Syntax**

```
TNBnnnn.Debug [:= Boolean];
```

The following are possible values for Debug property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Enables debug information.</td>
</tr>
<tr>
<td>False</td>
<td>Disables debug information.</td>
</tr>
</tbody>
</table>

**Remarks**

By default, the debug file is saved in the application directory. You can change the directory by setting the `DebugDir` property.

Default value is **False**.

**See also** `DebugDir` property.
4.2.2.1.2.7 DebugDir

Specifies the directory for debug files.

**Visual Basic Syntax**

\[
TNBnnnn.DebugDir [= String]
\]

**Delphi Syntax**

\[
TNBnnnn.DebugDir [:= String];
\]

**Remarks**

By default, the debug file is saved in the application's directory. You can change it by setting this property to another directory. This directory must exists, otherwise, no debug information will be generated and an error will not occur.

The default value is the application directory.

*See also* [Debug](#) property.

4.2.2.1.2.8 DeviceName

Sets an specific telnet device name defined in the telnet server.

**Visual Basic Syntax**

\[
TNBnnnn.DeviceName [= String]
\]

**Delphi Syntax**

\[
TNBnnnn.DeviceName [:= String];
\]

**Remarks**

If no name is given, the telnet server will assign an available device from a public pool in case of it exists.

*See also* [UserId](#) and [Password](#) properties.
4.2.2.1.2.9 EditFields

Returns an array containing the editable screen fields.

**Visual Basic Syntax**

\[
[TNBField =] TNBnnnn.EditFields (Index As Variant)
\]

**Delphi Syntax**

\[
[TNBField :=] TNBnnnn.EditFields [Index: variant];
\]

**Remarks**

This list includes only the unprotected *TNBFields* of the mainframe's screen. If you specified a wrong index value, a null value is returned.

See also *HostFields* property, *TNBFields* and *TNBField* Classes.

4.2.2.1.2.10 ExcludeEmptyFields

Specifies to the telnet component to exclude or not host empty fields.

**Visual Basic Syntax**

\[
TNBnnnn.ExcludeEmptyFields [= Boolean]
\]

**Delphi Syntax**

\[
TNBnnnn.ExcludeEmptyFields [= Boolean];
\]

**Remarks**

Default value is False.

See also *HostFields* property, *TNBFields* and *TNBField* Classes.

4.2.2.1.2.11 Extended

Specifies if the telnet 5250 extended protocol is allowed by this client.

**Visual Basic Syntax**
4.2.2.1.2.12 Host

Sets/gets the TCP/IP address or DNS name of the Host to connect to. It can be a name like 'myhost.mydomain.com' or directly an IP address such as '206.155.164.20'.

**Visual Basic Syntax**

\[TNBnnnn.Host \ = \ String\]

**Delphi Syntax**

\[TNBnnnn.Host \ := \ String\];

**Remarks**

The host name stored in the Host property must be alpha numeric, and can't contain spaces nor special characters (i.e. : / ? \)

**See also** Port property, Connect method and OnConnect event.
[**TNBField**] = **TNBnnnn.HostFields** *(Index As Variant)*

**Delphi Syntax**

```delphi
[TNBField := TNBnnnn.HostFields [index: variant];
```

**Remarks**

This list includes all **TNBFields** (protected and unprotected) of the mainframe's screen. If you specified a wrong index value, a null value is returned.

See also **EditFields** property, **TNBFields** and **TNBField** classes.

---

4.2.2.1.2.14 IsExtended

Returns a boolean value indicating if the current connection uses TN5250E capabilities.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBnnnn.IsExtended
```

**Delphi Syntax**

```delphi
[Boolean := TNBnnnn.IsExtended;
```

See also **Extended** property.

---

4.2.2.1.2.15 KeepAlive

Allows to enable a keep-alive mechanism used by some telnet servers.

**Visual Basic Syntax**

```vbnet
TNBnnnn.KeepAlive [= Boolean]
```

**Delphi Syntax**

```delphi
TNBnnnn.KeepAlive [= Boolean];
```

**Remarks**
Default value for telnet 3270 server (IBM S/3XX mainframes) is **True**.
Default value for telnet 5250 server (IBM AS/400 mainframes) is **False**.

### 4.2.2.1.2.16 LastErrMsg

Returns a string that specifies the last communication error reported by the host in the telnet current connection.

**Visual Basic Syntax**

```
[String =] TNBnnnn.LastErrMsg
```

**Delphi Syntax**

```
[String :=] TNBnnnn.LastErrMsg;
```

### 4.2.2.1.2.17 Password

Sets a Password for the UserId specified.

**Visual Basic Syntax**

```
TNBnnnn.Password [= String]
```

**Delphi Syntax**

```
TNBnnnn.Password [:= String];
```

See also **DeviceName** and **UserId** properties.

### 4.2.2.1.2.18 Port

TCP Port of the TN3270/TN5250 host to connect to.

**Visual Basic Syntax**

```
TNBnnnn.Port [= Integer]
```
**Delphi Syntax**

```delphi
TNBnnnn.Port [= Integer];
```

**Remarks**

By default it takes the standard telnet port. You can change it if the non standard port is to be used.

Default value is **23**.

See also **Host** property, **Connect** method and **OnConnect** event.

---

**4.2.2.1.2.19 Rows**

Return the total number of rows of the current terminal type.

**Visual Basic Syntax**

```vbnet
[Integer =] TNBnnnn.Rows
```

**Delphi Syntax**

```delphi
[Integer :=] TNBnnnn.Rows;
```

**Remarks**

Depending on the terminal type chosen, this property returns the values 24, 32 or 43.

See also **Cols** and **TerminalType** property.

---

**4.2.2.1.2.20 ScreenRow**

Returns a string contained in the specified row.

**Visual Basic Syntax**

```vbnet
[String =] TNBnnnn.ScreenRow (Row As Integer)
```

**Delphi Syntax**
[String :=] TNBnnnn.ScreenRow [Row: integer];

**Remarks**

If you specified a wrong row value, a range error will occur.

See also HostFields property.

### 4.2.2.1.2.21 SessionState

Returns the current session state.

**Visual Basic Syntax**

```vb
[TNBSS =] TNBnnnn.SessionState
```

**Delphi Syntax**

```delphi
[TNBSS :=] TNBnnnn.SessionState;
```

It can return any of TNBSessionState constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssNoSession</td>
<td>Not in session.</td>
</tr>
<tr>
<td>ssSSCPLU</td>
<td>In session with VTAM.</td>
</tr>
<tr>
<td>ssLULU</td>
<td>In session with VTAM Application.</td>
</tr>
</tbody>
</table>

**Remarks**

While the telnet connection is closed this property returns always ssNoSession value. When the telnet connection is opened it can return:

- ssNoSession, indicating that the connection was established at telnet level but not at 5250 Data Stream level.
- ssSSCPLU, indicating that a connection with the VTAM was established, but there's not a session with an final application (like CICS, TSO, etc.). (This state is only valid for TN3270 Hosts).
- ssLULU, indicating that the connection with the VTAM application (like CICS, TSO, etc.) has been established.

See also TNBSS type.
4.2.2.1.2.22 SubKey

Sets/gets the subkey under which will be stored the connection profiles.

Visual Basic Syntax

TNBnnnn.SubKey [= String]

Delphi Syntax

TNBnnnn.SubKey [= String];

Remarks

This property is only valid if the TnbProfiles property has been assigned to a TnbProfiles component.

Default value is "TNB3270E" for TNB3270 component and "TNB5250" for TNB5250 component.

See also TnbProfiles property and TnbProfiles component.

4.2.2.1.2.23 TerminalType

Sets the terminal type.

Visual Basic Syntax

TNBnnnn.TerminalType [= TB5250TT]

Delphi Syntax

TNBnnnn.TerminalType [= TB5250TT];

It can take any of the following constant values:

For 5250:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt5211m2 = 0</td>
<td>IBM-5211-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3179m2 = 1</td>
<td>IBM-3179-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3477mFC = 2</td>
<td>IBM-3477-FC 27 rows x 132 columns</td>
</tr>
<tr>
<td>tt3180m2 = 3</td>
<td>IBM-3180-2 27 rows x 132 columns</td>
</tr>
</tbody>
</table>

Visual Basic Syntax
**TNBnnnnX.** **TerminalType** [ = **TNB3270TT**]

**Delphi Syntax**

**TNBnnnnX.** **TerminalType**; [: = **TNB3270TT**]

**For 3270:**

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt3278m2 = 0</td>
<td>IBM-3278-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m2E = 1</td>
<td>IBM-3278-2-E 24 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m3 = 2</td>
<td>IBM-3278-3 32 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m3E = 3</td>
<td>IBM-3278-3-E 32 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m4 = 4</td>
<td>IBM-3278-4 43 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m4E = 5</td>
<td>IBM-3278-4-E 43 rows x 80 columns extended</td>
</tr>
</tbody>
</table>

See also **TNB5250TT** and **TNB3270TT** constants.

---

4.2.2.1.2.24 **TnbProfiles**

Sets the **TnbProfiles** component as the profile manager for this component.

**Visual Basic Syntax**

**TNBnnnn.** **TnbProfiles** [ = **TnbProfiles**]

**Delphi Syntax**

**TNBnnnn.** **TnbProfiles** [ := **TnbProfiles**];

**Remarks**

After setting this property, you can access to a collection of connections to be generated and customized through the **ShowEditor** method.

See Also **TnbProfiles** component, **Subkey** property and **ShowEditor** method.
4.2.2.1.2.25 TnbTrace

Sets the TNBTrace component as its trace agent.

**Visual Basic Syntax**

```
TNBnnnn.TnbTrace [= TNBTrace]
```

**Delphi Syntax**

```
TNBnnnn.TnbTrace [:= TNBTrace];
```

**Remarks**

You must set this property to get trace information from telnet component.

**See Also** [TNBTrace](#) component.

---

4.2.2.1.2.26 UserId

Sets a UserId for the device name specified.

**Visual Basic Syntax**

```
TNBnnnn.UserId [= String]
```

**Delphi Syntax**

```
TNBnnnn.UserId [:= String];
```

**See also** [DeviceName](#) and [Password](#) properties.

---

4.2.2.1.2.27 XLockDelay

Controls the delay in milliseconds between the last [OnScreenChange](#) and [OnSystemUnlock](#) events.

**Visual Basic Syntax**

```
TNBnnnn.XLockDelay [= Integer]
```

**Delphi Syntax**
4.2.2.1.3 Methods

4.2.2.1.3.1 About

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```vbnet
TNBnnnn.About
```

**Delphi Syntax**

```delphi
TNBnnnn.About;
```

4.2.2.1.3.2 ClassNameIs

Indicates if a specified name is the corresponding class name.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBnnnn.ClassNameIs (Name As String)
```

**Delphi Syntax**

```delphi
[Boolean :=] TNBnnnn.ClassNameIs (Name: string);
```

4.2.2.1.3.3 CloseEditor

Closes the modal dialog with the connection properties.

**Visual Basic Syntax**

```vbnet
TNBEmulator.CloseEditor
```

**Delphi Syntax**

```delphi
TNBEmulator.CloseEditor;
```
See also TnbProfiles control, TnbProfiles property and ShowEditor method.

4.2.2.1.3.4 Connect

The Connect method establishes the connection to the Telnet server. The Host and Port properties must be assigned for the connection to be successfully established.

Visual Basic Syntax

TNBnnnn.Connect

Delphi Syntax

TNBnnnn.Connect;

Remarks

If the specified host doesn't exist or if the connection fails, the OnConnectionFail event is fired. If the connection is successfully established, the OnConnect event is fired.

See also Host and Port properties, Disconnect method, OnConnect and OnConnectionFail events.

4.2.2.1.3.5 Disconnect

The Disconnect method ends the connection with the Telnet server.

Visual Basic Syntax

TNBnnnn.Disconnect

Delphi Syntax

TNBnnnn.Disconnect;

Remarks

After the client disconnects, the OnDisconnect event is fired. If the client is not connected to any server, the disconnect method has no effect.

See also Connect method and OnDisconnect event.
4.2.2.1.3.6 FromPool

Gets a free Tnb3270/Tnb5250 object from a pool of objects.

**Delphi Syntax**

\[
\text{tnbxxxx."
FromPool"(poolId as String,[sessionId as String],[InactivityTimeout as Integer])}
\]

**Remarks**

Gets a Tn3270/Tn5250 object from a pool named "poolId". If there's no a free object in the named pool, it creates a new one, adds it to the pool and returns the newly created object.

If the sessionId parameter is specified, the specific associated object is returned. Under ASP.NET environment, each Telnet object has associated the Session's SessionId string. Under ASP.NET, calling this method passing the Sessions's SessionId as second parameter it is a safe way to get access to the same telnet object from one page to another under the same ASP.NET session.

InactivityTimeout parameter indicates, in minutes, how much time the telnet object will remain in the pool without activity. Default timeout is 5 minutes.

To return an object to the pool you must call Release method.

See also Release method.

4.2.2.1.3.7 GetScreenRowEx

GetScreenRowEx can be used to retrieve the text buffer of one row of the current screen. Additionally, you can retrieve the standard or extended attribute corresponding to each field.

**Visual Basic Syntax**

\[
[\text{String } = ] \text{TNBnnnn.GetScreenRowEx} (\text{Row As Integer}, [\text{Attr As Boolean } = \text{True}], [\text{Eab As Boolean } = \text{True}])
\]

**Delphi Syntax**

\[
[\text{String :=} ] \text{TNBnnnn.GetScreenRowEx} (\text{Row:integer}, [\text{Attr:boolean } = \text{True}], [\text{Eab:boolean } = \text{True}]);
\]
**Remarks**

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned string, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in an 80x24 display, each row returned will occupy 160 characters.

**See also** GetScreenText method.

---

### 4.2.2.1.3.8 GetScreenText

GetScreenText returns the text buffer of a portion of the current screen. You can choose to retrieve the attributes or the extended attributes of each field joined to the character data.

**Visual Basic Syntax**

```
[String =] TNBnnnn.GetScreenText (StartPos As Integer, EndPos As Integer, [Attr As Boolean = True], [Eab As Boolean = True])
```

**Delphi Syntax**

```
```

**Remarks**

StartPos and EndPos specify the start position and the end position of the text buffer that will be returned.

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned string, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in an 80x24 display, each row returned will occupy 160 characters.
See also GetScreenRowEx method.

4.2.2.1.3.9 IsConnected

Returns a boolean value indicating if a connection is currently established with the Host.

**Visual Basic Syntax**

```vb net
[Boolean =] TNBnnnn.IsConnected
```

**Delphi Syntax**

```delphi
[Boolean :=] TNBnnnn.IsConnected;
```

See also Connect and Disconnect methods.

4.2.2.1.3.10 LoadFromXMLFile

Loads the XML code corresponding to the TNBnnnn object.

**Visual Basic Syntax**

```vb net
TNBnnnn.LoadFromXMLFile (XMLFile as String)
```

**Delphi Syntax**

```delphi
TNBnnnn.LoadFromXMLFile (XMLFile: string);
```

4.2.2.1.3.11 Release

Returns a Tnb3270/Tnb5250 object to the pool of telnet objects.

**VB.NET Syntax**

```vb net
tnxxxx.Release
```

**C# Syntax**
`tnxxxx.Release();`

**Remarks**

Returns a Tnb3270/Tnb5250 object to the pool of telnet objects. Used in combination with `FromPool` method.

See also `FromPool` method.

### 4.2.2.1.3.12 SaveToXMLFile

Exports the XML code corresponding to the *TNBnnnn object*.

**Visual Basic Syntax**

`TNBnnnn.SaveToXMLFile (XMLFile As String)`

**Delphi Syntax**

`TNBnnnn.SaveToXMLFile (XMLFile: string);`

### 4.2.2.1.3.13 SendAid

Sets an Attention Identifier Key and sends the modified fields to the Mainframe. Modified fields can be input fields (unprotected) or sometimes protected fields, having the property Modified set to True.

**Visual Basic Syntax**

`TNBnnnn.SendAid (AidKey As string)`

**Delphi Syntax**

`TNBnnnn.SendAid (AidKey:string);`

This method works like the AidKey property, but it takes a string with the name of the AID key as a parameter. You can specify only one key for each call to the method.

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
</tbody>
</table>
Clear | CLEAR key
--- | ---
ATTN | Attention key
SysReq | System Request key
PF1 | PF01 key
PF2 | PF02 key
PF3 | PF03 key
PF4 | PF04 key
PF5 | PF05 key
PF6 | PF06 key
PF7 | PF07 key
PF8 | PF08 key
PF9 | PF09 key
PF10 | PF10 key
PF11 | PF11 key
PF12 | PF12 key
PF13 | PF13 key
PF14 | PF14 key
PF15 | PF15 key
PF16 | PF16 key
PF17 | PF17 key
PF18 | PF18 key
PF19 | PF19 key
PF20 | PF20 key
PF21 | PF21 key
PF22 | PF22 key
PF23 | PF23 key
PF24 | PF24 key
PA1 | PA1 key

See also AIDKey property.

4.2.2.1.3.14 SendKeys

SendKeys can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host functions.

**Visual Basic Syntax**

```vbnet
TNBnnnn.SendKeys (Keys As String)
```

**Delphi Syntax**

```delphi
TNBnnnn.SendKeys (Keys: string);
```

The following table lists the functions keys and its corresponding codes.
<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the Keys parameter:

"logon applid(cics)@E"

**Remarks**

The execution of the SendKeys is stopped when an AID Key is sent to the Host.

See also **AIDKey** property.

4.2.2.1.3.15 **ShowEditor**

Shows a modal dialog with the connection properties.

**Visual Basic Syntax**
**TNBnnnn.ShowEditor**

**Delphi Syntax**

`TNBnnnn.ShowEditor;`

**Remarks**

If you haven't assigned the TnbProfiles property, the connection profiles list will not be shown. Setting the **TnbProfiles** property to a valid **TNBProfiles**, allows you to define connection profiles at run time and save them into a file for future reuse.

**See also** **TnbProfiles** control and **TnbProfiles** property.

### 4.2.2.1.4 Events

#### 4.2.2.1.4.1 OnConnect

Occurs after a successfully connection to the server is established.

**See also** **Connect** method and, **OnConnectionFail** and **OnDisconnect** events.

#### 4.2.2.1.4.2 OnConnectionFail

Occurs after the connection to the server fails.

**See also** **Connect** method, **OnConnect** and **OnDisconnect** events.

#### 4.2.2.1.4.3 OnDisconnect

Occurs after a disconnection of the server.

**See also** **Disconnect** method.
4.2.2.1.4.4 OnScreenChange

Occurs after a new data screen has arrived.

See also OnSystemLock and OnSystemUnlock events.

4.2.2.1.4.5 OnSendAid

Occurs before an Aid key is to be sent.

Remarks

This event is fired when a SendAid or SendKeys methods or the AidKey property are used to send an Aid Key and data to the mainframe. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator component.

See also SendAid, SendKeys methods and AidKey property from telnet components, and Lock method from TNBFields Class.

4.2.2.1.4.6 OnSessionState

Occurs when a session-state change takes place.

Remarks

A session state can be ssNoSession, ssSSCPLU and ssLULU. Any transition between the states fires this event.

See also SessionState property.

4.2.2.1.4.7 OnSystemLock

Occurs when a terminal changes to a system locked state.

Remarks

During this state, the terminal is waiting for any response from the mainframe. Sending data isn't possible until the OnSystemUnlock event is fired.
4.2.2.1.4.8 OnSystemUnlock

Occurs when a terminal changes to a system unlocked state.

**Remarks**

Only during this state, the component can send data to the host system.

See also `OnSystemLock` and `OnScreenChange` events.

4.2.2.2 TTnb3287 Reference

4.2.2.2.1 TTnb3287 Component

**Properties**

- `AttachMode`
- `CodePage`
- `ConversionTable`
- `Debug`
- `DebugDir`
- `DeviceName`
- `Host`
- `IsConnected`
- `Port`
- `Printer`
- `SendToPrinter`
- `SubKey`
- `Text`
- `TnbProfiles`

**Methods**

- `Connect`
- `Disconnect`
- `ShowPrinterEditor`

**Events**

- `OnConnect`
- `OnConnectFail`
- `OnDisconnect`
4.2.2.2.1.1 Properties

Controls the type of association between the emulation session and the printing session.

**Visual Basic Syntax**

\[ TTnb3287.\text{AttachMode} [= TnbAttachMode] \]

**Delphi Syntax**

\[ TTnb3287.\text{AttachMode} [:= TnbAttachMode]; \]

**Remarks**

The following are possible values for AttachMode property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>amDirect</td>
<td>Enables direct association between sessions.</td>
</tr>
<tr>
<td>amAssociate</td>
<td>The host handle the association between sessions.</td>
</tr>
</tbody>
</table>

Default value is **amDirect**.

Specifies the internal ASCII-EBCDIC conversion table.

**Visual Basic Syntax**

\[ TTnb3287.\text{CodePage} [= TNBXCP] \]

**Delphi Syntax**

\[ TTnb3287.\text{CodePage} [:= TNBXCP]; \]

TNBXCP can take any of the following Code Pages constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia</td>
<td>037</td>
</tr>
<tr>
<td>cpBelgium</td>
<td>037</td>
</tr>
<tr>
<td>cpCanada</td>
<td>037</td>
</tr>
</tbody>
</table>
Specifies an external ASCII-EBCDIC conversion table to be loaded from a file. If no one is specified, an internal conversion table is used, according to `CodePage` property setting.

**Visual Basic Syntax**

`TTnb3287.ConversionTable [= String]`

**Delphi Syntax**

`TTnb3287.ConversionTable := String;`

**Remarks**

The specified file must exist and contain a valid conversion table format.

The format of conversion table is:

```
[Ascii-To-Ebcdic]
Ascii hexadecimal code 1 = Ebcdic hexadecimal code 1
Ascii hexadecimal code 2 = Ebcdic hexadecimal code 2
```
\[Ebcdic\] \[To-Ascii\]

\[
\text{Ebcdic hexadecimal code } n = \text{Ascii hexadecimal code } n
\]

See also \textbf{CodePage} property.

Allows the generation of trace information for debugging purposes. The debugging can help you to track down both, errors and logic information.

**Visual Basic Syntax**

\[
TTnb3287.Debug \ [ = \ Boolean]
\]

**Delphi Syntax**

\[
TTnb3287_Debug \ [ := \ Boolean];
\]

**Remarks**

The following are possible values for Debug property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Enables debug information.</td>
</tr>
<tr>
<td>False</td>
<td>Disables debug information.</td>
</tr>
</tbody>
</table>

Default value is \textbf{False}.

By default, the debug file is saved in the application directory. You can change the directory by setting the \textbf{DebugDir} property.

See also \textbf{DebugDir} property.

Specifies the directory for debug files.

**Visual Basic Syntax**
TTnb3287. **DebugDir [= String]**

**Delphi Syntax**

```
TTnb3287. DebugDir [:= String];
```

**Remarks**

By default, the debug file is saved in the application's directory. You can change it by setting this property to another directory. This directory must exists, otherwise, no debug information will be generated and an error will not occur. The default value is the application directory.

See also **Debug** property.

Sets an specific telnet device name defined in the telnet server.

**Visual Basic Syntax**

```
TTnb3287. DeviceName [= String]
```

**Delphi Syntax**

```
TTnb3287. DeviceName [:= String];
```

**Remarks**

When AttachMode is set to amDirect, this component will try to connect to the specified Device Name (LU). When AttachMode is set to amAssociate, the telnet server will try to assign a device associated to the specified LU.

See also **Host** and **AttachMode** properties.

TCP/IP address or DNS name of the Host to connect to. It can be a name like 'myhost.mydomain.com' or directly an IP address such as '206.155.164.20'.

**Visual Basic Syntax**

```
TTnb3287. Host [= String]
```

**Delphi Syntax**
**TTnb3287.Host [:= String];**

**Remarks**

The host name stored in the Host property must be alpha numeric, and can't contain spaces nor special characters (i.e. : " / \)

See also **Port** property, **Connect** method and **OnConnect** event.

Returns a boolean value indicating if a connection is currently established with the Host.

**Visual Basic Syntax**

```vbnet
[Boolean =>] Tnb3287.IsConnected
```

**Delphi Syntax**

```delphi
```

See also **Connect** and **Disconnect** methods.

Sets/gets the TCP Port of the TN3270/TN5250 host to connect to.

**Visual Basic Syntax**

```vbnet
TTnb3287.Port [= Integer]
```

**Delphi Syntax**

```delphi
TTnb3287.Port [:= Integer];
```

**Remarks**

By default it takes the standard telnet port. You can change it if the non standard port is to be used.
Default value is **23**.

See also **Host** property, **Connect** method and **OnConnect** event.
Points to the TTnbPrinter object which control the printer activity.

**Visual Basic Syntax**

```vbnet
[TTnbPrinter =] TTnb3287.Printer
```

**Delphi Syntax**

```delphi
[TTnbPrinter :=] TTnb3287.Printer;
```

Controls the capability of sending a document to a printer.

**Visual Basic Syntax**

```
TTnb3287.SendToPrinter [= Boolean]
```

**Delphi Syntax**

```
TTnb3287.SendToPrinter [= Boolean];
```

Sets/gets the subkey under which it will be stored the connection profiles.

**Visual Basic Syntax**

```
TTnb3287.SubKey [= String]
```

**Delphi Syntax**

```
TTnb3287.SubKey [= String];
```

**Remarks**

This property is only valid if the **TNBProfiles** property is being used.

**See also** **TNBProfiles** Control.
Returns the received printer-data.

**Visual Basic Syntax**

```
[String =] TTNb3287.Text
```

**Delphi Syntax**

```
[String :=] TTNb3287.Text;
```

**Remarks**

When a **OnDataAvailable** event is fired, Text property returns the text data just received. Once the **OnEndDocument** event is fired, the Text property holds the whole received printer-data.

Sets the TTnbProfiles control as the profile manager for this control.

**Visual Basic Syntax**

```
TTnb3287.TnbProfiles [= TTnbProfiles]
```

**Delphi Syntax**

```
TTnb3287.TnbProfiles [= TTnbProfiles];
```

**See Also** **Subkey** property.

### 4.2.2.2.1.2 Methods

The Connect method establishes the connection to the Telnet server. The **Host** and **Port** properties must be assigned for the connection to be successfully established.

**Visual Basic Syntax**

```
TTnb3287.Connect
```

**Delphi Syntax**

`
TTnb3287.Connect;

**Remarks**

If the specified host doesn't exist or if the connection fails, the [OnConnectionFail](#) event is fired. If the connection is successfully established, the [OnConnect](#) event is fired.

**See also** [Host](#) and [Port](#) properties, [Disconnect](#) method, [OnConnect](#) and [OnConnectionFail](#) events.

The Disconnect method ends the connection with the Telnet server.

### Visual Basic Syntax

`TTnb3287.Disconnect`

### Delphi Syntax

`TTnb3287.Disconnect;`

**Remarks**

After the client disconnects, the [OnDisconnect](#) event is fired. If the client is not connected to any server, the disconnect method has no effect.

**See also** [Connect](#) method and [OnDisconnect](#) event.

Shows the printer's configuration dialog.

### Visual Basic Syntax

`TTnb3287.ShowPrinterEditor`

### Delphi Syntax

`TTnb3287.ShowPrinterEditor;`
4.2.2.1.3 Events

Occurs after a successfully connection to the server is established.

See also Connect method and, OnConnectionFail and OnDisconnect events.

Occurs after the connection to the server fails.

See also Connect method, OnConnect and OnDisconnect events.

Occurs after a data file buffer is received.

See also Text property, OnStartDocument and OnEndDocument events.

Occurs after a disconnection of the server.

See also Disconnect method.

Occurs after the last data file buffer is received.

See also OnStartDocument event.

Occurs after the first data file buffer is received.

See also OnEndDocument event.
4.2.2.3 TTnb3812 Reference

4.2.2.3.1 TTnb3812 Component

**Properties**

- CodePage
- Debug
- DebugDir
- DeviceName
- Host
- IsConnected
- MsgQueueLib
- MsgQueueName
- Port
- Printer
- SendToPrinter
- SubKey
- Text
- TnbProfiles

**Methods**

- Connect
- Disconnect
- ShowPrinterEditor

**Events**

- OnConnect
- OnConnectFail
- OnDisconnect
- OnEndDocument
- OnDataAvailable
- OnStartDocument

4.2.2.3.1.1 Properties

Specifies the internal ASCII-EBCDIC conversion table.

**Visual Basic Syntax**

```
TTnb3812.CodePage = TNBCP
```

**Delphi Syntax**
TTnb3812.CodePage [:= TNBCP];

TNBCP can take any of the following Code Pages constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia = 037</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium = 037</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada = 037</td>
<td>Canada</td>
</tr>
<tr>
<td>cpCzechRepublic = 870</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cpDenmark = 277</td>
<td>Denmark</td>
</tr>
<tr>
<td>cpFinland = 278</td>
<td>Finland</td>
</tr>
<tr>
<td>cpFrance = 297</td>
<td>France</td>
</tr>
<tr>
<td>cpGermany = 273</td>
<td>Germany</td>
</tr>
<tr>
<td>cpGreece = 423</td>
<td>Greece</td>
</tr>
<tr>
<td>cpGreece2 = 875</td>
<td>Greece</td>
</tr>
<tr>
<td>cpItaly = 280</td>
<td>Italy</td>
</tr>
<tr>
<td>cpLatinAmerica = 284</td>
<td>Latin America</td>
</tr>
<tr>
<td>cpNetherlands = 037</td>
<td>Netherlands</td>
</tr>
<tr>
<td>cpNorway = 277</td>
<td>Norway</td>
</tr>
<tr>
<td>cpPoland = 870</td>
<td>Poland</td>
</tr>
<tr>
<td>cpPortugal = 037</td>
<td>Portugal</td>
</tr>
<tr>
<td>cpSpain = 284</td>
<td>Spain</td>
</tr>
<tr>
<td>spSweden = 278</td>
<td>Sweden</td>
</tr>
<tr>
<td>cpSwitzerlandFrench = 500</td>
<td>Switzerland (French)</td>
</tr>
<tr>
<td>cpSwitzerlandGerman = 500</td>
<td>Switzerland (German)</td>
</tr>
<tr>
<td>cpUnitedKingdom = 285</td>
<td>United Kingdom</td>
</tr>
<tr>
<td>cpUnitedStates = 037</td>
<td>United States</td>
</tr>
<tr>
<td>cpInternational = 500</td>
<td>International</td>
</tr>
</tbody>
</table>

Default cpUnitedStates.

See also ConversionTable property and Code Pages constants.

Specifies an external ASCII-EBCDIC conversion table to be loaded from a file. If no one is specified, an internal conversion table is used, according to CodePage property setting.

**Visual Basic Syntax**

```
TTnb3812.ConversionTable [= String]
```

**Delphi Syntax**

```
TTnb3812.ConversionTable [:= String];
```
Remarks

The specified file must exist and contain a valid conversion table format.

The format of conversion table is:

[Ascii-To-Ebcdic]
Ascii hexadecimal code 1 = Ebcdic hexadecimal code 1
Ascii hexadecimal code 2 = Ebcdic hexadecimal code 2
... ...
Ascii hexadecimal code n = Ebcdic hexadecimal code n

[Ebcdic-To-Ascii]
Ebcdic hexadecimal code 1 = Ascii hexadecimal code 1
Ebcdic hexadecimal code 2 = Ascii hexadecimal code 2
... ...
Ebcdic hexadecimal code n = Ascii hexadecimal code n

See also CodePage property.

Visual Basic Syntax

TTnb3812.Debug [= Boolean]

Delphi Syntax

TTnb3812.Debug [= Boolean];

Remarks

The following are possible values for Debug property:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>Enables debug information.</td>
</tr>
<tr>
<td>False</td>
<td>Disables debug information.</td>
</tr>
</tbody>
</table>

Default value is False.

By default, the debug file is saved in the application directory. You can change the directory by setting the DebugDir property.

See also DebugDir property.
Specifies the directory for debug files.

**Visual Basic Syntax**

```
TTnb3812.DebugDir [= String]
```

**Delphi Syntax**

```
TTnb3812.DebugDir [:= String];
```

**Remarks**

By default, the debug file is saved in the application's directory. You can change it by setting this property to another directory. This directory must exists, otherwise, no debug information will be generated and an error will not occur.

The default value is the application directory.

**See also** [Debug](#) property.

Sets/gets an specific telnet device name defined in the telnet server.

**Visual Basic Syntax**

```
TTnb3812.DeviceName [= String]
```

**Delphi Syntax**

```
TTnb3812.DeviceName [:= String];
```

**Remarks**

If no name is given, the telnet server will assign an available device from a public pool in case of it exists.

**See also** [Host](#) property.

Sets/gets the TCP/IP address or DNS name of the Host to connect to. It can be a name like 'myhost.mydomain.com' or directly an IP address such as '206.155.164.20'.

**Visual Basic Syntax**

```
```
TTnb3812.Host [= String]

**Delphi Syntax**

TTnb3812.Host [:= String];

**Remarks**

The host name stored in the Host property must be alpha numeric, and can't contain spaces nor special characters (i.e. : " / ?)

See also Port property, Connect method and OnConnect event.

Returns a boolean value indicating if a connection is currently established with the Host.

**Visual Basic Syntax**

[Boolean =] Tnb3812.IsConnected

**Delphi Syntax**


See also Connect and Disconnect methods.

Holds the library that contains the message used by the print writer for operational messages. The default value is '*LIBL'.

**Visual Basic Syntax**

TTnb3812.MsgQueueLib [= String]

**Delphi Syntax**

TTnb3812.MsgQueueLib [:= String];

See also MsgQueueName property.
Holds the name of the message queue used by the print writer for operational messages. The default value is 'QSYSOPR'.

**Visual Basic Syntax**

```
TTnb3812.MsgQueueName [= String]
```

**Delphi Syntax**

```
TTnb3812.MsgQueueName := String;
```

See also [MsgQueueLib](#) property.

Sets/gets the TCP Port of the TN3270/TN5250 host to connect to.

**Visual Basic Syntax**

```
TTnb3812.Port [= Integer]
```

**Delphi Syntax**

```
TTnb3812.Port := Integer;
```

**Remarks**

By default it takes the standard telnet port. You can change it if the non standard port is to be used.
Default value is 23.

See also [Host](#) property, [Connect](#) method and [OnConnect](#) event.

Points to the TTnbPrinter object that extends the TPrinter standard Delphi object to control the printer activity.

**Visual Basic Syntax**
[TTnbPrinter =] TTnb3812.Printer

**Delphi Syntax**

[TTnbPrinter :=] TTnb3812.Printer;

**See also** SendToPrinter property.

Controls the capability of send a document to a printer.

**Visual Basic Syntax**

TTnb3812.SendToPrinter [= Boolean]

**Delphi Syntax**

TTnb3812.SendToPrinter [= Boolean];

**See also** Printer property.

Sets/gets the subkey under which will be stored the connection profiles.

**Visual Basic Syntax**

TTnb3812.SubKey [= String]

**Delphi Syntax**

TTnb3812.SubKey [= String];

**Remarks**

This property is only valid if the TNBProfiles property is being used.

**See also** TNBProfiles Control.
Returns the received printer-data.

**Visual Basic Syntax**

```vbnet
[String =] TTnb3812.Text
```

**Delphi Syntax**

```delphi
[String :=] TTnb3812.Text;
```

**Remarks**

When a **OnDataAvailable** event is fired, Text property returns the text data just received. Once the **OnEndDocument** event is fired, the Text property holds the whole received printer-data.

Sets the TTnbProfiles control as the profile manager for this control.

**Visual Basic Syntax**

```vbnet
TTnb3812.TnbProfiles [= TTnbProfiles]
```

**Delphi Syntax**

```delphi
TTnb3812.TnbProfiles [:= TTnbProfiles];
```

**See Also** **Subkey** property.

### 4.2.2.3.1.2 Methods

The Connect method establishes the connection to the Telnet server. The **Host** and **Port** properties must be assigned for the connection to be successfully established.

**Visual Basic Syntax**

```vbnet
TTnb3812.Connect
```

**Delphi Syntax**

```delphi
TTnb3812.Connect;
```
Remarks

If the specified host doesn't exist or if the connection fails, the `OnConnectionFail` event is fired. If the connection is successfully established, the `OnConnect` event is fired.

See also `Host` and `Port` properties, `Disconnect` method, `OnConnect` and `OnConnectionFail` events.

The `Disconnect` method ends the connection with the Telnet server.

**Visual Basic Syntax**

```
TTnb3812.Disconnect
```

**Delphi Syntax**

```
TTnb3812.Disconnect;
```

Remarks

After the client disconnects, the `OnDisconnect` event is fired. If the client is not connected to any server, the disconnect method has no effect.

See also `Connect` method and `OnDisconnect` event.

Shows the printer's configuration dialog.

**Visual Basic Syntax**

```
TTnb3812.ShowPrinterEditor
```

**Delphi Syntax**

```
TTnb3812.ShowPrinterEditor;
```
4.2.2.3.1.3 Events

Occurs after a successfully connection to the server is established.

See also **Connect** method and, **OnConnectionFail** and **OnDisconnect** events.

Occurs after the connection to the server fails.

See also **Connect** method, **OnConnect** and **OnDisconnect** events.

Occurs after a data file buffer is received.

See also **Text** property, **OnStartDocument** and **OnEndDocument** events.

Occurs after a disconnection of the server.

See also **Disconnect** method.

Occurs after the last data file buffer is received.

See also **OnStartDocument** event.

Occurs after the first data file buffer is received.

See also **OnEndDocument** event.
4.2.2.4 TTnbField Class Reference

4.2.2.4.1 TTnbField Class

Implements a Screen field object.

Properties

- Attr
- AutoEnter
- Blinking
- ByPass
- Col
- Color
- Data
- Eab
- Editable
- High
- Len
- Mandatory
- Modified
- MonoCase
- Name
- Numeric
- Pos
- Reverse
- Row
- Skip
- Underline
- Visible
- HllApiAttr
- HllApiEab

4.2.2.4.2 Properties

4.2.2.4.2.1 Attr

Sets/gets the attribute of a TTNBField.

Visual Basic Syntax

```
TNBField.Attr [= Integer]
```

Delphi Syntax

```
TNBField.Attr [:= Integer];
```
Remarks

This property is for reserved use.

4.2.2.4.2.2 AutoEnter

Returns true, if an Enter AID Key should be sent when the focus of the field gets lost.

**Visual Basic Syntax**

```vba
[Boolean =] TNBField.AutoEnter
```

**Delphi Syntax**

```delphi
[Boolean :=] TNBField.AutoEnter;
```

Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

4.2.2.4.2.3 Blinking

Returns true if the field should be shown blinking.

**Visual Basic Syntax**

```vba
[Boolean :=] TNBField.Blinking
```

**Delphi Syntax**

```delphi
```

Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also High, Reverse and Underline properties.
4.2.2.4.2.4  ByPass

Returns true for unprotected fields that shouldn't gain the keyboard focus.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBField.ByPass
```

**Delphi Syntax**

```delphi
```

**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program. This property is valid only for unprotected fields.

4.2.2.4.2.5  Col

Returns the screen column coordinate where the field begins.

**Visual Basic Syntax**

```vbnet
TNBField.Col [Integer =]
```

**Delphi Syntax**

```delphi
TNBField.Col [Integer :=];
```

**Remarks**

Valid values are from 1 to `TNBnnnnX.Cols` property.

**See Also** `Row`, `Pos` and `Len` properties.
4.2.2.4.2.6 Color

Returns an integer that represents the default color corresponding to the standard or extended field attribute.

**Visual Basic Syntax**

```vbnet
[Integer =] TNBField.Color
```

**Delphi Syntax**

```delphi
[Integer :=] TNBField.Color;
```

**Remarks**

This property returns an integer value and it's a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

*See Also* Blinking, High, Reverse and Underline properties.

4.2.2.4.2.7 Data

Sets/gets and gets the data of the field.

**Visual Basic Syntax**

```vbnet
TNBField.Data [= String]
```

**Delphi Syntax**

```delphi
TNBField.Data [:= String];
```

**Remarks**

Changing the data of an unprotected field sets the Modified property to true.

*See Also* Modified property.
4.2.2.4.2.8 Eab

Sets/gets the extended attribute of a TTNBField.

**Visual Basic Syntax**

```
[Bytee =] TNBField.Eab
```

**Delphi Syntax**

```
[Bytee :=] TNBField.Eab;
```

**Remarks**

This property is for reserved use.

4.2.2.4.2.9 Editable

Returns true if the field is editable (unprotected), otherwise returns false.

**Visual Basic Syntax**

```
[Boolean =] TNBField.Editable
```

**Delphi Syntax**

```
[Boolean :=] TNBField.Editable;
```

See Also **Mandatory** property.

4.2.2.4.2.10 High

Returns true if the field should be shown with a high intensity color or bold, otherwise returns false.

**Visual Basic Syntax**

```
[Boolean =] TNBField.High
```

**Delphi Syntax**

```
[Boolean :=] TNBField.High;
```
Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, Reverse and Underline properties.

4.2.2.4.2.11 Len

Sets/gets the field's length.

Visual Basic Syntax

`TNBField.Len [Integer =]`

Delphi Syntax

`TNBField.Len [Integer :=];`

See Also Row, Col and Pos properties.

4.2.2.4.2.12 Mandatory

Returns true if this field is editable and must be filled in.

Visual Basic Syntax

`[Boolean =] TNBField.Mandatory`

Delphi Syntax

`[Boolean :=] TNBField.Mandatory;`

Remarks

This property is valid only for unprotected fields.

See Also Editable property.
### 4.2.2.4.2.13 Modified

This property indicates if the field has been modified.

**Visual Basic Syntax**

```vbnet
TNBField.Modified [= Boolean]
```

**Delphi Syntax**

```delphi
TNBField.Modified [= Boolean];
```

**See Also** [Data](#) property.

### 4.2.2.4.2.14 MonoCase

This property indicates if the field accepts uppercase characters only.

**Visual Basic Syntax**

```vbnet
[Boolean =] TNBField.MonoCase
```

**Delphi Syntax**

```delphi
[Boolean :=] TNBField.MonoCase;
```

**Remarks**

This property is valid only for unprotected fields.

**See Also** [Numeric](#) property.

### 4.2.2.4.2.15 Name

Sets/gets a string with the field name. This name is made by the telnet components and it can be modified before first use.
**Visual Basic Syntax**

`TNBField.Name [= String]`

**Delphi Syntax**

`TNBField.Name [:= String]`;

**Remarks**

The string returned is the result of a concatenation between "R", the value of Row property, "C" and the value of Col property. For example, the name of a field located at row 4 column 30 will be R4C30.

---

**4.2.2.4.2.16 Numeric**

Indicates if the field accepts numeric digits only.

**Visual Basic Syntax**

`[Boolean =] TNBField.Numeric`

**Delphi Syntax**


**Remarks**

This property is valid only for unprotected fields.

*See Also* Monocase property.

---

**4.2.2.4.2.17 Pos**

Sets/gets the field position in the screen.

**Visual Basic Syntax**

`TNBField.Pos [= Integer]`
Delphi Syntax

TNBField.Pos [:= Integer];

Remarks

This property returns an integer ranging from 1 to TNBnnnn.Cols property multiplied by TNBnnnn.Rows property. The order of the numeration is from the left to the right and from top to bottom.

See Also Col, Row and Len properties.

4.2.2.4.2.18 Reverse

Returns true if the field should be shown in reverse video and false if the field should be shown in normal video.

Visual Basic Syntax

[Boolean =] TNBField.Reverse

Delphi Syntax

[Boolean :=] TNBField.Reverse;

Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, High and Underline property.

4.2.2.4.2.19 Row

Sets/gets the screen row coordinate where the field begins.

Visual Basic Syntax

TNBField.Row [= Integer]

Delphi Syntax
TNField.Row := Integer;

Remarks
Valid values are from 1 to TNBnnnn.Rows property.

See Also Col, Len and Pos properties.

4.2.2.4.2.20 Skip
Returns true for fields that shouldn't gain the keyboard focus.

Visual Basic Syntax
[Boolean =] TNField.Skip

Delphi Syntax
[Boolean :=] TNField.Skip;

Remarks
When the property returns True, the cursor will skip this field preventing it to be written. Normally, you should consider this property if you are going to implement a terminal emulation program.

4.2.2.4.2.21 Underline
Returns true if the field should be shown underlined and false if the field should not be shown underlined.

Visual Basic Syntax
[Boolean =] TNField.Underline

Delphi Syntax
[Boolean :=] TNField.Underline;
Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, High and Reverse properties.

4.2.2.4.2.22 Visible

Returns true if the field is visible, otherwise it returns false.

Visual Basic Syntax

[Boolean =] TNBField. Visible

Delphi Syntax


Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program. If the field is a label (protected) and visible is false, it shouldn't be shown. If the field is editable and visible is false, it means that this is a password field.

4.2.2.4.2.23 HllApiAttr

Returns the attribute of TTNBField in HLLAPI-compliance format.

Visual Basic Syntax

[Byte =] TNBField. HllApiAttr

Delphi Syntax

[Byte :=] TNBField. HllApiAttr;

Remarks

This is a terminal-emulation behavior attribute. Normally, you should consider this
property if you are going to implement a terminal emulation program.

See Also Editable, Blinking, Color, High and Reverse properties.

4.2.2.4.2.24 HllApiEab

Returns the extended attribute of TTNBFIELD in HLLAPI-compliance format.

**Visual Basic Syntax**

```
[Byte =] TTNBFIELD.HllApiEab
```

**Delphi Syntax**

```
[Byte :=] TTNBFIELD.HllApiEab;
```

**Remarks**

This is a terminal-emulation behavior attribute. Normally, you should consider this property if you are going to implement a terminal emulation program.

See Also Blinking, Color, High and Reverse properties.

4.2.2.5 TTNbfFields Class Reference

4.2.2.5.1 TTNbfFields Class

Implements the Screen Fields list.

**Properties**

- Count
- CursorField
- CursorPos
- Items

**Methods**

- Lock
4.2.2.5.2 Properties

4.2.2.5.2.1 Count

Returns the total number of TNBFields in the collection.

**Visual Basic Syntax**

\[
\text{[INTEGER = ] TNBFields.Count}
\]

**Delphi Syntax**

\[
\text{[INTEGER :=] TNBFields.Count;}
\]

See also Items property.

4.2.2.5.2.2 CursorField

Returns the field where the cursor is located.

**Visual Basic Syntax**

\[
\text{[TNBField = ] TNBFields.CursorField}
\]

**Delphi Syntax**

\[
\text{[TNBField :=] TNBFields.CursorField;}
\]

See also HostFields and EditFields property from TNB5250/TNB3270 class and CursorPos property.

4.2.2.5.2.3 CursorPos

Gets/sets the host-screen cursor position.

**Visual Basic Syntax**

\[
\text{TNBFields.CursorPos [= INTEGER]}
\]

**Delphi Syntax**
TNBFields.CursorPos := Integer;

Remarks
Valid cursor position goes from 1 to the total number of rows multiplied by the total number of columns available for the terminal type specified. The current cursor position is available in both HostFields and EditFields properties from TNB5250/TNB3270 class. To set a new cursor position you must do it on EditFields property.

See also HostFields and EditFields property from TNB5250/TNB3270 class.

4.2.2.5.2.4 Items
Contains a collection of TNBField objects.

Visual Basic Syntax
[TNBField =] TNB3270Fields.Items(Index As Variant)

Delphi Syntax
[TNBField :=] TNB3270Fields.Items[Index:Variant];

Remarks
Use Items to get an specific field from the array. The Index parameter indicates the object’s index in the collection, where 0 is the index of the first element and (Count - 1) is the index of the last element.
Also, you can access an specific TNBField by its field name.
Use Items with the Count property to iterate through all the objects in the list.

See also Count and Name properties from TNBField class.

4.2.2.5.3 Methods
4.2.2.5.3.1 Lock
Used to prevent the filling of EditFields after an OnSendAid event.

Visual Basic Syntax
TNBFields.Lock

**Delphi Syntax**

```
TNBFields.Lock;
```

**Remarks**

It is valid only for `EditFields` property from `TNB5250/TNB3270` class.

See also `OnSendAid` event from `TNB5250/TNB3270` class.

---

### 4.2.2.6 TTNBIndFile Reference

#### 4.2.2.6.1 TTNBIndFile Component

**Properties**

- `AbortString`
- `Append`
- `Ascii`
- `BlkSize`
- `BufSize`
- `Bytes`
- `CrLf`
- `Direction`
- `FtCommand`
- `FtMode`
- `FtName`
- `HostFileName`
- `HostKind`
- `LocalFileName`
- `Lrecl`
- `PrimarySpace`
- `RecFm`
- `SecondarySpace`
- `ShowDialog`
- `ShowEditor`
- `Stream`
- `SubKey`
- `TimeOut`
- `TnbCom`
- `TnbProfiles`
- `Units`
Events

- OnAborting
- OnComplete
- OnRunning
- OnUpdateLength

4.2.2.6.2 Properties

4.2.2.6.2.1 AbortString

This property contains a strings that describes why the file transfer was aborted.

**Visual Basic Syntax**

[String =] _TNBIndFile_.AbortString

**Delphi Syntax**

[String :=] _TNBIndFile_.AbortString;

See also Abort method.

4.2.2.6.2.2 Append

If set to True, the file to be transferred is appended to the existing one. If the destination file doesn't exist (HostFileName in case of Send method or LocalFileName in case of Receive method), it is treated as new.
If set to False, no append is taken place. If the destination file exists, it is replaced.

**Visual Basic Syntax**

_TNBIndFile_.Append [ = Boolean]

**Delphi Syntax**
4.2.2.6.2.3 Ascii

It is used when transferring files from host to PC to convert EBCDIC characters to Ascii characters.

**Visual Basic Syntax**

```
TNBIndFile.AscII := Boolean
```

**Delphi Syntax**

```
TNBIndFile.AscII; [:= Boolean]
```

*See also* [Crlf](#) property.

4.2.2.6.2.4 BlkSize

Sets/gets the block size to be used for the host destination file ([HostFileName](#) property) when a new file is created in [Send](#) method or [Run](#) method with [Direction](#) := 0 indicator set.

Must be a multiple of record length ([Lrec](#) property), if fixed blocked ([RecFm](#) property) file format is specified.

**Visual Basic Syntax**

```
TNBIndFile.BlkSize [= Integer]
```

**Delphi Syntax**

```
TNBIndFile.BlkSize; [:= Integer]
```

*See also* [HostFileName](#) property, [Lrec](#) property, [RecFm](#) property, [Run](#) method, [Send](#) method.
4.2.2.6.2.5 BufSize

This property has the size of the buffer used to do the file transfer. It's only used when the connection mode is DFT.

**Visual Basic Syntax**

```vbs
TNBIndFile.BufSize [= Integer]
```

**Delphi Syntax**

```delphi
TNBIndFile.BufSize; [= Integer]
```

See also *Stream* property, *Bytes* property.

4.2.2.6.2.6 Bytes

Is the current bytes transferred to the destination file, `HostFileName` in *Send* or *Run* (Direction:=0) methods, or `LocalFileName` in *Receive* or *Run* (Direction:=1) methods.

**Visual Basic Syntax**

```vbs
TNBIndFile.Bytes [Integer =]
```

**Delphi Syntax**

```delphi
TNBIndFile.Bytes; [Integer :=]
```

See also *OnUpdateLength* event.

4.2.2.6.2.7 CrLf

It is used when transferring file from a host to a PC.
If set to True, CRLF characters, chr(13)+chr(10), are added to the end of each line. If set to False no characters are added.

**Visual Basic Syntax**

```vbs
TNBIndFile.CrLf [= Boolean]
```
**Delphi Syntax**

`TNBIndFile.CrLf [:= boolean];`

See also `Ascii` property.

---

### 4.2.2.6.2.8 Direction

This property is used in conjunction with the `Run` method to indicate the transfer direction of the File Transfer. A 0 value indicates a send direction (from `LocalFileName` file name to `HostFileName` file name). A 1 value indicates a receive direction (from `HostFileName` file name to `LocalFileName` file name).

**Visual Basic Syntax**

`TNBIndFile.Direction [ = Integer]`

**Delphi Syntax**

`TNBIndFile.Direction; [:= Integer]`

See also `Run` method.

---

### 4.2.2.6.2.9 FtCommand

Sets/gets the name of the file transfer command in the host system, usually 'IND$FILE'. This property doesn't have to be changed from its default value.

**Visual Basic Syntax**

`TNBIndFile.FtCommand [ = IND$FILE]`

**Delphi Syntax**

`TNBIndFile.FtCommand; [:= IND$FILE]`
4.2.2.6.2.10 FtMode

Indicates the File Transfer Mode. By default the value is ftDFT, but if the host can't handle DFT mode it automatically changes to ftCut.

**Visual Basic Syntax**

[String =] TNBIndFile.FtMode

**Delphi Syntax**

[String :=] TNBIndFile.FtMode;

**Remarks**

Possible values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftDFT</td>
<td>DFT mode</td>
</tr>
<tr>
<td>ftCut</td>
<td>Cut mode</td>
</tr>
</tbody>
</table>

See also [TFTMode](#) constants.

4.2.2.6.2.11 FtName

It's the name of the file transfer session.

**Visual Basic Syntax**

TNBIndFile.FtName [= String]

**Delphi Syntax**

TNBIndFile.FtName; [:= String]

See also [TnbProfiles](#) property, [SubKey](#) property.

4.2.2.6.2.12 HostFileName

Sets/gets the name in the host system of the file to be transferred. It is used either in the Send / Receive methods mode or the Run method mode. The file name syntax has to
be valid for the host system.

**Visual Basic Syntax**

\[TNBIndFile.HostFileName\] \([= FileName.dat]\]

**Delphi Syntax**

\[TNBIndFile.HostFileName; \(\<![= FileName.dat]\]]

**Example**:

- 'File Name a1' (file name, file type, file mode)
  - valid name for VM/CMS.
- 'sys1.vtamlst(atcstr00)' valid name for PDS member name under Tso.
- 'sys2.user.dat' valid name for catalogued file.

See also **LocalFileName** property.

---

### 4.2.2.6.2.13 HostKind

Sets/gets the environment that the host supports.

**Visual Basic Syntax**

\[TNBIndFile.HostKind\] \([= THostKind]\]

**Delphi Syntax**

\[TNBIndFile.HostKind; \(\<![= THostKind]\]]

**Remarks**

Possible values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hVM</td>
<td>VM</td>
</tr>
<tr>
<td>hTSO</td>
<td>TSO</td>
</tr>
<tr>
<td>hCics</td>
<td>Cics</td>
</tr>
</tbody>
</table>

See also **THostKind** constants.
4.2.2.6.2.14 LocalFileName

Sets/gets the name in the local system of the file to be transferred. It is used either in the Send / Receive methods mode or the Run method mode. The file name syntax has to be valid in the local system.

Example:

Visual Basic Syntax

TNBIndFile.LocalFileName [= FileName.dat]

Delphi Syntax

TNBIndFile.LocalFileName; [:= FileName.dat]

See also HostFileName property.

4.2.2.6.2.15 LRecl

Sets/gets the record length to be used for the host destination file (HostFileName property) when a new file is created during the execution of a Send method or Run method (with Direction:=0 indicator set).
It must be a submultiple of block size (BlkSize property), if fixed blocked (RecFm property) file format is specified.

Visual Basic Syntax

TNBIndFile.LRecl [= Integer]

Delphi Syntax

TNBIndFile.LRecl; [:= Integer]

See also HostFileName property, BlkSize property, RecFm property, Run method, Send method.
4.2.2.6.2.16 PrimSpace

Sets/gets the primary space quantity used to allocate the host `HostFileName` file in the `Units` selected, when a new file is created during the execution of a `Send` method or `Run` method (with `Direction`:0 indicator set).

**Visual Basic Syntax**

`TNBIndFile.PrimSpace [= Integer]`

**Delphi Syntax**

`TNBIndFile.PrimSpace; [:= Integer]`

See also `HostFileName` property, `SecSpace` property, `BlkSize` property, `RecFm` property, `Units` property, `Run` method, `Send` method.

4.2.2.6.2.17 RecFm

Sets/gets the record format of the host `HostFileName` file.

**Visual Basic Syntax**

`TNBIndFile.RecFm [= Integer]`

**Delphi Syntax**

`TNBIndFile.RecFm; [:= Integer]`

Possible values for this property are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default file format</td>
</tr>
<tr>
<td>1</td>
<td>Fixed file format</td>
</tr>
<tr>
<td>2</td>
<td>Variable file format</td>
</tr>
<tr>
<td>3</td>
<td>Undefined file format</td>
</tr>
</tbody>
</table>

See also `BlkSize` property, `HostFileName` property, `PrimSpace` property, `Run` method, `SecSpace` property, `Send` method, `Units` property.
4.2.2.6.2.18 SecSpace

Sets/gets the secondary space quantity used to allocate the host HostFileName file in the Units selected, when a new file is created during the execution of a Send method or Run method (with Direction:=0 Send indicator set).

**Visual Basic Syntax**

```vbnet
TNBIndFile.SecSpace [= Integer]
```

**Delphi Syntax**

```delphi
TNBIndFile.SecSpace; [= Integer]
```

See also HostFileName property, PrimSpace property, BlkSize property, RecFm property, Units property, Run method, Send method.

4.2.2.6.2.19 ShowDialog

Specifies if the file transfer 'Bytes Transferred' dialog appears in each buffer sent to the destination. This dialog could be used to Cancel the file transfer in progress.

**Visual Basic Syntax**

```vbnet
TNBIndFile.ShowDialog [= Boolean]
```

**Delphi Syntax**

```delphi
TNBIndFile.ShowDialog; [= Boolean]
```

See also OnAborting event.

4.2.2.6.2.20 ShowEditor

Specifies if the File Transfer Settings dialog appears previously to each file transfer request to let the user modify the file transmission parameters. All the fields in both Transfers and Advanced tabs have the corresponding property that
can be set programmatically instead than doing it through this dialog.

<table>
<thead>
<tr>
<th>ActionFieldData</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background</td>
<td>Color</td>
</tr>
<tr>
<td>Description</td>
<td>EndCol</td>
</tr>
<tr>
<td>EndRow</td>
<td>Id</td>
</tr>
<tr>
<td>Pattern</td>
<td>StartCol</td>
</tr>
<tr>
<td>StartRow</td>
<td>ViewAs</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

```
TNBIndFile.ShowEditor [= Boolean]
```

**Delphi Syntax**

```
TNBIndFile.ShowEditor; [= Boolean]
```

4.2.2.6.2.21 Stream

Sets/gets a byte stream that contains the information the PC received from host or the information the PC will send to the host.

For more information about streams read Delphi Help about TStream.

**Visual Basic Syntax**

```
TNBIndFile.Stream (Buffer As String, BufSize As Integer)
```

**Delphi Syntax**

```
TNBIndFile.Stream (Buffer:string; BufSize:integer);
```

See also **BufSize** property, **Bytes** property.
4.2.2.6.2.22 SubKey

Sets/gets the subkey under which the connection profiles will be stored.

**Visual Basic Syntax**

`TNBIndFile.SubKey [= String]`

**Delphi Syntax**

`TNBIndFile.SubKey; [:= String]`

**Remarks**

Default value is 'TNB3270E' for TTNB3270E Component and 'TNB5250' for TTNB5250 Component.

See also `TnbProfiles` property, `FtName` property.

4.2.2.6.2.23 TimeOut

Sets/gets the time slice to wait for the host connection to be established.

**Visual Basic Syntax**

`TNBIndFile.TimeOut [= Integer]`

**Delphi Syntax**

`TNBIndFile.TimeOut; [:= Integer]`

See also `Receive` method, `Run` method, `Send` method.

4.2.2.6.2.24 TnbCom

Sets/gets the TTNB3270E or TTNB5250 Component as the telnet client Component.

**Visual Basic Syntax**

`TNBIndFile.TnbCom [= TNBnnnnX]`
**Delphi Syntax**

```
TNBIndFile.TnbCom; [= TNBnnnnX]
```

### 4.2.2.6.2.25 TnbProfiles

Sets/gets the TnbProfiles Component as the profile manager for this Component.

**Visual Basic Syntax**

```
TNBIndFile.TnbProfiles [= TnbProfiles]
```

**Delphi Syntax**

```
TNBIndFile.TnbProfiles; [= TnbProfiles]
```

After setting this property, you can access to a collection of connections to be generated and customized through the `ShowEditor` method.

*See also TnbProfiles reference, SubKey property, ShowEditor property.*

### 4.2.2.6.2.26 Units

Sets/gets the type of allocation unit of the host `HostFileName` file used when a new file is created during the execution of a `Send` method or `Run` method (with `Direction`:=0 Send indicator set).

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Default unit of allocation</td>
</tr>
<tr>
<td>1</td>
<td>Tracks unit of allocation</td>
</tr>
<tr>
<td>2</td>
<td>Cylinder unit of allocation</td>
</tr>
<tr>
<td>3</td>
<td>Average Blocks unit of allocation</td>
</tr>
</tbody>
</table>

**Visual Basic Syntax**

```
TNBIndFile.Units [= Integer]
```

**Delphi Syntax**

```
TNBIndFile.Units; [= Integer]
```
See also **BlkSize** property, **PrimSpace** property, **RecFm** property, **SecSpace** property.

### 4.2.2.6.3 Methods

#### 4.2.2.6.3.1 Abort

The transfer is interrupted after changing the current transfer's status to abort.

**Visual Basic Syntax**

```vbnet
TNBIndFile.Abort
```

**Delphi Syntax**

```delphi
TNBIndFile.Abort;
```

See also **AbortString** property, **OnAborting** event.

#### 4.2.2.6.3.2 LoadFromXMLFile

Loads the XML code corresponding to the `TNBIndFile` object.

**Visual Basic Syntax**

```vbnet
TNBIndFile.LoadFromXMLFile (XMLFile As String)
```

**Delphi Syntax**

```delphi
TNBIndFile.LoadFromXMLFile (XMLFile: string);
```

See also **SaveToXMLFile** method.

#### 4.2.2.6.3.3 Receive

This method is used to receive into **LocalFileName** file the **HostFileName** file.

**Visual Basic Syntax**
4.2.2.6.3.4 Run

This method is used to send LocalFileName file from the PC to the HostFileName in the host if Direction property is set to 0=Send, or vice versa if Direction property is set to 1=Receive.

Visual Basic Syntax

TNBIndFile.Run

Delphi Syntax

TNBIndFile.Run;

See also Direction property, Receive method, Send method.

4.2.2.6.3.5 SaveToXMLFile

Exports the XML code corresponding to the TNBIndFile object.

Visual Basic Syntax

TNBIndFile.SaveToXMLFile (XMLFile As String)

Delphi Syntax

TNBIndFile.SaveToXMLFile (XMLFile: string);

See also LoadFromXMLFile method.
4.2.2.6.3.6  Send

This method is used to send LocalFileName file from the PC to the HostFileName in the host using the properties directly modified or using the Editor dialog to fill them interactively.

**Visual Basic Syntax**

TNBindFile.Send

**Delphi Syntax**

TNBindFile.Send;

See also Direction property, Run method, Receive method.

4.2.2.6.4  Events

4.2.2.6.4.1  OnAborting

Occurs after the Cancel button in the 'Bytes Transferred' dialog is pressed indicating a user defined cancel operation.

**Declaration**

procedure OnAborting (Sender: TObject)

**Parameters**

Sender: The TNBindFile object that raises the event.

See also OnComplete event.

4.2.2.6.4.2  OnComplete

Occurs after the file transfer operation is finished.

**Declaration**

procedure OnComplete (Sender: TObject)

**Parameters**
Sender: The \texttt{TNBIndFile} object that raises the event.

\textbf{See also} \texttt{OnAborting} event.

\subsection{4.2.2.6.4.3 OnRunning}

Occurs when the file transfer operation has been started.

\textbf{Declaration}

\begin{verbatim}
procedure OnRunning (Sender: TObject)
\end{verbatim}

\textbf{Parameters}

Sender: The \texttt{TNBIndFile} object that raises the event.

\textbf{See also} \texttt{OnComplete} event.

\subsection{4.2.2.6.4.4 OnUpdateLength}

Occurs every time a buffer of data of \texttt{BufSize} bytes are sent from the source file to the destination file. Previously this event is fired the \texttt{Bytes} property is updated accordingly to reflect the total amount of data sent.

\textbf{Declaration}

\begin{verbatim}
procedure OnUpdateLength (Sender: TObject)
\end{verbatim}

\textbf{Parameters}

Sender: The \texttt{TNBIndFile} object that raises the event.

\textbf{See also} \texttt{OnRunning} event.
4.2.2.6.5 Constants

4.2.2.6.5.1 TFtMode

These values are used in the `FTMode` property of `TtnbIndFile` control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftDFT</td>
<td>DFT mode</td>
</tr>
<tr>
<td>ftCut</td>
<td>Cut mode</td>
</tr>
</tbody>
</table>

4.2.2.6.5.2 THostKind

These values are used in the `HostKind` property of `TtnbIndFile` control.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hVM</td>
<td>VM</td>
</tr>
<tr>
<td>hTSO</td>
<td>TSO</td>
</tr>
<tr>
<td>hCics</td>
<td>Cics</td>
</tr>
</tbody>
</table>

4.2.2.7 Constants

4.2.2.7.1 TNBCP

These values are used in the `CodePage` property of telnet components.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpAustralia = 037</td>
<td>Australian</td>
</tr>
<tr>
<td>cpBelgium = 037</td>
<td>Belgium</td>
</tr>
<tr>
<td>cpCanada = 037</td>
<td>Canada</td>
</tr>
<tr>
<td>cpCzechRepublic = 870</td>
<td>Czech Republic</td>
</tr>
<tr>
<td>cpDenmark = 277</td>
<td>Denmark</td>
</tr>
<tr>
<td>cpFinland = 278</td>
<td>Finland</td>
</tr>
<tr>
<td>cpFrance = 297</td>
<td>France</td>
</tr>
<tr>
<td>cpGermany = 273</td>
<td>Germany</td>
</tr>
<tr>
<td>cpGreece = 423</td>
<td>Greece</td>
</tr>
<tr>
<td>cpGreece2 = 875</td>
<td>Greece</td>
</tr>
<tr>
<td>cpItaly = 280</td>
<td>Italy</td>
</tr>
<tr>
<td>cpLatinAmerica = 284</td>
<td>Latin America</td>
</tr>
<tr>
<td>cpNetherlands = 037</td>
<td>Netherlands</td>
</tr>
<tr>
<td>cpNorway = 277</td>
<td>Norway</td>
</tr>
<tr>
<td>cpPoland = 870</td>
<td>Poland</td>
</tr>
<tr>
<td>cpPortugal = 037</td>
<td>Portugal</td>
</tr>
</tbody>
</table>
4.2.2.7.2 TNBAid

These values are used in the AidKey property of telnet components.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>akPF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>akPF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>akPF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>akPF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>akPF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>akPF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>akPF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>akPF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>akPF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>akPF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>akPF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>akPF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>akPF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>akPF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>akPF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>akPF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>akPF24</td>
<td>PF24 key</td>
</tr>
</tbody>
</table>
4.2.2.7.3  TNBSS

These values are used in the **SessionState** property of telnet components.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssNoSession</td>
<td>Not in session.</td>
</tr>
<tr>
<td>ssSSCPLU</td>
<td>In session with VTAM.</td>
</tr>
<tr>
<td>ssLULU</td>
<td>In session with VTAM Application.</td>
</tr>
</tbody>
</table>

4.2.2.7.4  TNB5250TT

These values are used in the **TerminalType** property of TNB5250 component.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt5211m2 = 0</td>
<td>IBM-5211-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3179m2 = 1</td>
<td>IBM-3179-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3477mFC = 2</td>
<td>IBM-3477-FC 27 rows x 132 columns</td>
</tr>
<tr>
<td>tt3180m2 = 3</td>
<td>IBM-3180-2 27 rows x 132 columns</td>
</tr>
</tbody>
</table>

4.2.2.7.5  TNB3270TT

These values are used in the **TerminalType** property of TNB3270 component.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt3278m2 = 0</td>
<td>IBM-3278-2 24 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m2E = 1</td>
<td>IBM-3278-2-E 24 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m3 = 2</td>
<td>IBM-3278-3 32 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m3E = 3</td>
<td>IBM-3278-3-E 32 rows x 80 columns extended</td>
</tr>
<tr>
<td>tt3278m4 = 4</td>
<td>IBM-3278-4 43 rows x 80 columns</td>
</tr>
<tr>
<td>tt3278m4E = 5</td>
<td>IBM-3278-4-E 43 rows x 80 columns extended</td>
</tr>
</tbody>
</table>
4.2.3 Terminal Emulator Components

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack Delphi Terminal Emulator Components.

4.2.3.1 TTnbEmulator Component

4.2.3.1.1 TTnbEmulator Component

This is an implementation of the screen-emulation panel as a component without the keyboard interface.

Properties

- Ctl3d
- CursorPos
- DisableKbdLockOnError
- EnableSelection
- ErrorLine
- FontAutoSize
- InputInhibitEnabled
- InsertMode
- IsInputInhibited
- LeftMargin
- RulerType
- StyleName
- SubKey
- TnbAidHook
- TnbCom
- TnbHotSpots
- TnbProfiles
- TopMargin
- Visible

Methods

- About
- AidKeyPressed
- AsVariant
- ClassNameIs
- Copy
- GetScreenRowEx
- GetScreenText
- LoadFromXMLFile
- LocalKeyPressed
- Paste
- PrintScreen
- PutFields
- Refresh
- SaveToXMLFile
- SendKeys
- ShowEditor
Events

- **OnClick**
- **OnCursorPos**
- **OnDblClick**
- **OnInputInhibitChange**
- **OnInsertModeChange**

Constants

- **TTnbRulerType**

4.2.3.1.2 Properties

4.2.3.1.2.1 Ctl3d

Sets/gets the 3D control's appearance.

**Visual Basic Syntax**

```
TNBEmulator.Ctl3d [= Boolean]
```

**Delphi Syntax**

```
TNBEmulator.Ctl3d [:= Boolean];
```

Default value is **False**

4.2.3.1.2.2 CursorPos

Gets/sets the host-screen cursor position.

**Visual Basic Syntax**

```
TNBEmulator.CursorPos [= Integer]
```

**Delphi Syntax**

```
TNBEmulator.CursorPos [:= Integer];
```

Remarks
Valid cursor position goes from 1 to the total number of rows multiplied by the total number of columns available for the terminal type specified. The current cursor position is available in both `HostFields` and `EditFields` properties from `TTnb5250/TTnb3270` class. To set a new cursor position you must do it on `EditFields` property.

See also `HostFields` and `EditFields` property from `TTnb5250/TTnb3270` class.

### 4.2.3.1.2.3 DisableKbdLockOnError
Set.gets a boolean value to lock/unlock the keyboard when the host returns an error message.

**Visual Basic Syntax**

```vbnet
TNBEmulator.DisableKbdLockOnError [= Boolean]
```

**Delphi Syntax**

```delphi
TNBEmulator.DisableKbdLockOnError [= Boolean];
```

**Remarks**

Default value is `False`.

See Also `ErrorLine` property.

### 4.2.3.1.2.4 EnableSelection
Enables/disables the selection of an emulation-screen area for copying it to the clipboard.

**Visual Basic Syntax**

```vbnet
TNBEmulator.EnableSelection [= Boolean]
```

**Delphi Syntax**

```delphi
TNBEmulator.EnableSelection [= Boolean];
```

**Remarks**

It must set to true if you plan to use `Copy` method, to allow the selection of a copy.
area.

Default value is True.

See Also Copy and Paste methods.

### 4.2.3.1.2.5 ErrorLine

Set/gets the line number where the error message will be displayed.

**Visual Basic Syntax**

`TNBXEmulator.ErrorLine [= Integer]`

**Delphi Syntax**

`TNBXEmulator.ErrorLine := Integer;`

**Remarks**

Default value is 0.

See Also DisableKbdLockOnError property.

### 4.2.3.1.2.6 FontAutoSize

Set/gets the autosize state of the emulator-screen font.

**Visual Basic Syntax**

`TNBEmulator.FontAutoSize [= Boolean]`

**Delphi Syntax**

`TNBEmulator.FontAutoSize := Boolean;`

**Remarks**

Default value is True.
4.2.3.1.2.7 InputInhibitEnabled

Enables/disables the input inhibit condition when any key is pressed on a protected field.

**Visual Basic Syntax**

```vbnet
TNBEmulator.InputInhibitEnabled [= Boolean]
```

**Delphi Syntax**

```delphi
TNBEmulator.InputInhibitEnabled [: = Boolean];
```

**Remarks**

Default value is **True**.

*See Also* [IsInputInhibited](#) property and [OnInputInhibitChange](#) event.

4.2.3.1.2.8 InsertMode

Sets/gets the insert/overwrite keyboard mode for typing on unprotected fields.

**Visual Basic Syntax**

```vbnet
TNBEmulator.InsertMode [= Boolean]
```

**Delphi Syntax**

```delphi
TNBEmulator.InsertMode [: = Boolean];
```

**Remarks**

Default value is **False**.

*See Also* [OnInsertModeChange](#) event.

4.2.3.1.2.9 IsInputInhibited

Returns true if the input of the emulator-screen is inhibited.
4.2.3.1.2.10 LeftMargin

Sets emulation screen left margin.

**Visual Basic Syntax**

```vbnet
TNBEmulator.LeftMargin [= Integer]
```

**Delphi Syntax**

```delphi
TNBEmulator.LeftMargin [= Integer];
```

**Remarks**

Left margin emulation value is measured in pixels.

4.2.3.1.2.11 RulerType

Sets/gets the format style of the ruler. This ruler shows the cursor position.

**Visual Basic Syntax**

```vbnet
TNBEmulator.RulerType [= TTnbRulerType]
```

**Delphi Syntax**

```delphi
TNBEmulator.RulerType [= TTnbRulerType];
```

**Remarks**
For example, **ruCrosshair** constant value will draw one vertical and one horizontal lines, and the intersection will be the cursor position.

Valid constant values are:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtNone = 0</td>
<td>No lines</td>
</tr>
<tr>
<td>rtCrosshair = 1</td>
<td>Horizontal and vertical lines</td>
</tr>
<tr>
<td>rtVertical = 2</td>
<td>Only vertical line</td>
</tr>
<tr>
<td>rtHorizontal = 3</td>
<td>Only horizontal line</td>
</tr>
</tbody>
</table>

*See Also* **CursorPos** property and **TtnbRulerType** constants.

### 4.2.3.1.2.12 StyleName

Sets/gets the Style from the collection of styles saved through the **TNBProfiles** component.

**Visual Basic Syntax**

```vbnet
TNBEmulator.StyleName [= String]
```

**Delphi Syntax**

```delphi
TNBEmulator.StyleName := String;
```

**Remarks**

This property is valid if a **TnbProfiles** property has been previously assigned.

*See Also* **TNBProfiles** component and **ShowEditor** method.

### 4.2.3.1.2.13 SubKey

Sets/gets the subkey under which the screen styles will be stored.

**Visual Basic Syntax**

```vbnet
TNBEmulator.SubKey [= String]
```

**Delphi Syntax**

```delphi
TNBEmulator.SubKey := String;
```
TNBEmulator.SubKey := String;

Remarks

This property is valid only if a TnbProfiles property has been assigned to a TNBProfiles component.

See also TnbProfiles property, SubKey and TNBProfiles component.

4.2.3.1.2.14 TnbAidHook

Sets the TNBAidHook component as the keyboard handler for this control.

Visual Basic Syntax

TNBEmulator.TnbAidHook( := TNBAidHook)

Delphi Syntax

TNBEmulator.TnbAidHook[ := TNBAidHook];

Remarks

Normally, to send information to the host through the TNBEmulator you must call the PutFields method and then send the properly AID key to the telnet component (see AidKey property or SendAid method of TNB3270/TNB5250 components). Assigning this property to a TNBAidHook component, you don't need to do these calls. When the TNBEmulator receives a function key from TNBAidHook, process it internally, sending the modified fields and the AID key pressed to the host.

See Also TNBAidHook component.

4.2.3.1.2.15 TnbCom

Sets the TNB3270 or TNB5250 component as the telnet client component.

Visual Basic Syntax

TNBEmulator.TnbCom( := TNBnnnn)

Delphi Syntax
\texttt{TNBEEmulator.TnbCom} \[= \text{TNBnnnn}\];

**Remarks**

You must set this property for TNBEEmulator control to allow it to work properly.

---

### 4.2.3.1.2.16 TnbHotSpots

Sets the \texttt{TNBHotSpot} component as the hotspots handler for this control.

**Visual Basic Syntax**

\texttt{TNBEEmulator.TnbHotSpots} \[= \text{TNBHotSpot}\]

**Delphi Syntax**

\texttt{TNBEEmulator.TnbHotSpots} \[= \text{TNBHotSpot}\];

**See Also** \texttt{TNBHotSpot} component.

---

### 4.2.3.1.2.17 TnbProfiles

Sets the component as the profile manager for this control.

**Visual Basic Syntax**

\texttt{TNBEEmulator.TnbProfiles} \[= \text{TNBProfiles}\]

**Delphi Syntax**

\texttt{TNBEEmulator.TnbProfiles} \[= \text{TNBProfiles}\];

**Remarks**

After setting this property, you can access to a collection of styles to be generated and customized through the \texttt{ShowEditor} method.

**See Also** \texttt{TNBProfiles} component, \texttt{StyleName} property and \texttt{ShowEditor} method.
4.2.3.1.2.18 TopMargin

Sets emulation screen top margin.

**Visual Basic Syntax**

TNBEmulator.TopMargin [= Integer]

**Delphi Syntax**

TNBEmulator.TopMargin [= Integer];

**Remarks**

Top margin emulation value is measured in pixels.

4.2.3.1.2.19 Visible

Sets/gets the visibility of the control at run-time.

**Visual Basic Syntax**

TNBEmulator.Visible [= Boolean]

**Delphi Syntax**

TNBEmulator.Visible [= Boolean];

**Remarks**

Default value is True.

4.2.3.1.3 Methods

4.2.3.1.3.1 About

Shows Integration Pack about dialog box.

**Visual Basic Syntax**
### 4.2.3.1.3.3 AsVariant

Returns the control as a variant data type variable.

**Visual Basic Syntax**

```
[Variant =] TNBEmulator.AsVariant
```

**Delphi Syntax**

```
[Variant :=] TNBEmulator.AsVariant;
```

### 4.2.3.1.3.2 AidKeyPressed

AidKeyPressed can be used to send AID keys to the host.

**Visual Basic Syntax**

```
TNBEmulator.AidKeyPressed (AidKey As String)
```

**Delphi Syntax**

```
TNBEmulator.AidKeyPressed (AidKey:string);
```

**Remarks**

The AID key go directly to the mainframe's host application. Doesn't remains local.

See also AIDKey and LocalKeyPressed properties, SendKeys and SendAid methods, OnAidKey event.
4.2.3.1.3.4 ClassNameIs
Indicates if a specified name is the corresponding class name.

**Visual Basic Syntax**

```
[Boolean =] TNBEmulator.ClassNameIs (Name As String)
```

**Delphi Syntax**

```
[Boolean :=] TNBEmulator.ClassNameIs (Name: string);
```

4.2.3.1.3.5 Copy
The Copy method copies the selected area to the clipboard.

**Visual Basic Syntax**

```
TNBEmulator.Copy
```

**Delphi Syntax**

```
TNBEmulator.Copy;
```

**Remarks**

The `EnableSelection` property must be set to true to allows you select an area of the emulator-screen.

*See also* `EnableSelection` property and `Paste` method.

4.2.3.1.3.6 GetScreenRowEx
GetScreenRowEx can be used to retrieve the text buffer of one row of the current screen. You can choose to recover the attribute or extended attribute of each field together to the character data.

**Visual Basic Syntax**

```
[String =] TNBEmulator.GetScreenRowEx (Row As Integer, [Attr As Boolean = True],
                                           [Eab As Boolean = True])
```

**Delphi Syntax**
[String :=] TNBEmulator.GetScreenRowEx (Row:integer; [Attr: boolean = True]; [Eab: boolean = True]);

**Remarks**

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in the row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned string, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

*See also* GetScreenText method.

### 4.2.3.1.3.7 GetScreenText

GetScreenText returns the text buffer of a portion of the current screen. You can choose to retrieve the attributes or the extended attributes of each field joined to the character data.

**Visual Basic Syntax**

[String :=] TNBEmulator.GetScreenText ([StartPos As Integer], [EndPos As Integer], [Attr As Boolean = True], [Eab As Boolean = True])

**Delphi Syntax**


**Remarks**

StartPos and EndPos specify the start position and the end position of the text buffer that will be returned.

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.
The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned string, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

See also GetScreenRowEx method.

4.2.3.1.3.8 LoadFromXMLFile

Loads the XML code corresponding to the TNBEmulator object.

Visual Basic Syntax

TNBEmulator.LoadFromXMLFile (XMLFile As String)

Delphi Syntax

TNBEmulator.LoadFromXMLFile (XMLFile: string);

4.2.3.1.3.9 LocalKeyPressed

LocalKeyPressed can be used to send AID keys to be executed in local mode.

Visual Basic Syntax

TNBEmulator.LocalKeyPressed (AidKey As String)

Delphi Syntax

TNBEmulator.LocalKeyPressed (AidKey:string);

Remarks

The AID key doesn't go to the mainframe's host application. Remains local.

See also AIDKey property, AidKeyPressed, SendKeys and SendAid methods, OnAidKey event.
4.2.3.1.3.10 Paste

The Paste method pastes text from the clipboard to the current cursor position.

**Visual Basic Syntax**

```
TNBEmulator.Paste
```

**Delphi Syntax**

```
TNBEmulator.Paste;
```

See also Copy method.

4.2.3.1.3.11 PrintScreen

Prints all the emulator-screen area.

**Visual Basic Syntax**

```
TNBEmulator.Print
```

**Delphi Syntax**

```
TNBEmulator.Print;
```

4.2.3.1.3.12 PutFields

Sends the modified fields to the **TnbCom** in use by TNBEmulator component.

**Visual Basic Syntax**

```
TNBEmulator.PutFields
```

**Delphi Syntax**

```
TNBEmulator.PutFields;
```

Remarks
To complete the data transmission to the host, you must set the AidKey property or use the SendAid method from the TnbCom component.

See also TnbCom property from TNBEmulator component, and, AidKey property and SendAid method from TNB3270/TNB5250 components.

4.2.3.1.3.13 Refresh

Refresh the fields for the current screen.

Visual Basic Syntax

TNBEmulator.Refresh

Delphi Syntax

TNBEmulator.Refresh;

4.2.3.1.3.14 SaveToXMLFile

Exports the XML code corresponding to the TNBEmulator object.

Visual Basic Syntax

TNBEmulator.SaveToXMLFile (XMLFile As String)

Delphi Syntax

TNBEmulator.SaveToXMLFile (XMLFile: string);

4.2.3.1.3.15 SendKeys

SendKeys can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host functions.

Visual Basic Syntax
TNBEmulator.SendKeys (Keys As String)

Delphi Syntax

TNBEmulator.SendKeys (Keys:string);

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the Keys parameter:

"logon applid(cics)@E"

Remarks
The execution of the SendKeys is stopped when an AID Key is sent to the Host.

See also AIDKey property and SendKeys method.

4.2.3.1.3.16 ShowEditor

Shows a modal dialog with the screen properties.

**Visual Basic Syntax**

```
TNBEmulator.ShowEditor
```

**Delphi Syntax**

```
TNBEmulator.ShowEditor;
```

**Remarks**

If you haven't assigned the TnbProfiles property, the style list will be shown disabled. Setting the TnbProfiles property to a valid TNBProfiles component, allows you to define screen-styles at run-time and save them into a file for future reuse.

See also TNBProfiles property.

4.2.3.1.4 Events

4.2.3.1.4.1 OnClick

Occurs when the user clicks the control.

See also OnDbClick event.

4.2.3.1.4.2 OnCursorPos

Occurs when the emulator cursor changes its position.
4.2.3.1.4.3 OnDblClick

Occurs when the user double-clicks the primary mouse button when the mouse pointer is over the control.

See also OnClick event.

4.2.3.1.4.4 OnInputInhibitChange

Occurs when the input from keyboard changes to inhibit or vice versa.

See also OnInsertModeChange event.

4.2.3.1.4.5 OnInsertModeChange

Occurs when the input from keyboard changes to insert mode or overwrite mode.

See also OnInputInhibitChange event.

4.2.3.1.5 Constants

4.2.3.1.5.1 TTnbRulerType

These values are used in the RulerType property of TNBEmulator component.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>rtNone = 0</td>
<td>No lines</td>
</tr>
<tr>
<td>rtCrosshair = 1</td>
<td>Horizontal and vertical lines</td>
</tr>
<tr>
<td>rtVertical = 2</td>
<td>Only vertical line</td>
</tr>
<tr>
<td>rtHorizontal = 3</td>
<td>Only horizontal line</td>
</tr>
</tbody>
</table>
4.2.3.2 TTnbAidHook Component

4.2.3.2.1 TTnbAidHook Component

This is an implementation of a keyboard interface, with mapping to the standard function keys of the 5250 and 3270 emulation.

**Properties**

- **Active**
- **HWND**
- **KbdFile**
- **KbdKind**
- **Mapping3270**
- **Mapping5250**
- **Profile**
- **SubKey**
- **TnbCom**
- **TnbKeyboard**
- **TnbProfiles**

**Methods**

- **ClassNameIs**
- **LoadFromXMLFile**
- **SaveToXMLFile**
- **ShowEditor**

**Events**

- **OnAidKey**

4.2.3.2.2 Properties

4.2.3.2.2.1 Active

Enables/disables the keyboard hook.

**Visual Basic Syntax**

```
TNBAidHook.Active [= Boolean]
```

**Delphi Syntax**

```
TNBAidHook.Active [:= Boolean];
```

**Remarks**
4.2.3.2.2.2 HWND

Sets/gets the handle to the window which the hook is being applied.

**Visual Basic Syntax**

TNBAidHook.HWND [= Integer]

**Delphi Syntax**

TNBAidHook.HWND := Integer;

**Remarks**

You can set the Keyboard Hook to be applied to any window. The OnAidKey event handler will be fired when a combination of keys in the mapping table is pressed.

**See also** OnAidKey event.

4.2.3.2.2.3 KdbFile

Allows you to specify an external file with a new keyboard mapping.

**Visual Basic Syntax**

TNBAidHook.KbdFile [= String]

**Delphi Syntax**

TNBAidHook.KbdFile [:= String];

**Remarks**

The file to be used can be exported from its property page or dialog box shown using the ShowEditor method.

**See Also** ShowEditor method.
4.2.3.2.2.4 KbdKind

Specify the keyboard mapping kind.

**Visual Basic Syntax**

```vb
TNBAidHook.KbdKind = TNBKbdKind
```

**Delphi Syntax**

```delphi
TNBAidHook.KbdKind := TNBKbdKind;
```

It can take one of this constants values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>k5250</td>
<td>5250 Keyboard</td>
</tr>
<tr>
<td>k3270</td>
<td>3270 Keyboard</td>
</tr>
</tbody>
</table>

**Remarks**

This component maintains separate keyboard mapping tables for each connection type. You can modify it through the property page associated to the component.

**See Also** `TNBKbdKind` constants.

4.2.3.2.2.5 Mapping3270

Sets/gets the keyboard mapping for TTnb3270 component.

**Visual Basic Syntax**

```vb
TNBAidHook.Mapping3270 = OleVariant
```

**Delphi Syntax**

```delphi
TNBAidHook.Mapping3270 := OleVariant;
```

**See Also** `Mapping5250` property.
4.2.3.2.2.6 Mapping5250

Sets/gets the keyboard mapping for TTnb5250 component.

**Visual Basic Syntax**

```
TNBAidHook.Mapping5250 [= OleVariant]
```

**Delphi Syntax**

```
TNBAidHook.Mapping5250 := OleVariant;
```

See Also  Mapping3270 property.

4.2.3.2.2.7 Profile

Sets/gets the name of the current keyboard map set.

**Visual Basic Syntax**

```
TNBAidHook.Profile [= String]
```

**Delphi Syntax**

```
TNBAidHook.Profile := String;
```

**Remarks**

The default keyboard map is called "Default". You can access a collection of keyboard maps to be generated and customized through the **ShowEditor** method. Then you can assign a different profile using **Profile** property. Different profiles generated will be stored through **TNBProfiles** component and the physical file will be identified through **KbdFile** property.

See Also  **TNBProfiles** component, **SubKey** and **KbdKind** properties and **ShowEditor** method.

4.2.3.2.2.8 SubKey

Sets/gets the subkey under which the keyboard maps will be stored.

**Visual Basic Syntax**
TNBAidHook.SubKey [= String]

**Delphi Syntax**

TNBAidHook.SubKey [= String];

**Remarks**

This property is valid only if a ProfilesControl property has been assigned to a TNBProfiles control.

See also TNBProfiles component, TnbProfiles and Profile properties and ShowEditor method.

---

### 4.2.3.2.2.9 TnbCom

Sets the TNB3270 or TNB5250 component as the telnet client component.

**Visual Basic Syntax**

TNBAidHook.TnbCom [= TNBnnnn]

**Delphi Syntax**

TNBAidHook.TnbCom [= TNBnnnn];

---

### 4.2.3.2.2.10 TnbKeyboard

Through this property you can access the collection of keyboard maps associated to TTnbAidHook component.

**Visual Basic Syntax**

TNBAidHook.TnbKeyboard [= TTnbCustomKeyBoardMap]

**Delphi Syntax**

TNBAidHook.TnbKeyboard [= TTnbCustomKeyBoardMap];

See also TNBProfiles component, ProfilesControl and Profile properties and ShowEditor method.
4.2.3.2.11 TnbProfiles

Sets the TnbProfiles Component as the profile manager for this control.

**Visual Basic Syntax**

```vbnet
TNBAidHook.TnbProfiles [= TnbProfiles]
```

**Delphi Syntax**

```delphi
TNBAidHook.TnbProfiles [: = TnbProfiles];
```

**Remarks**

After setting this property, you can access a collection of keyboard maps to be generated and customized through the ShowEditor method.

**See Also** TnbProfiles component, SubKey property and ShowEditor method.

4.2.3.2.3 Methods

4.2.3.2.3.1 ClassNameIs

Indicates if a specified name is the corresponding class name.

**Visual Basic Syntax**

```vbnet
(Boolean =) TNBAidHook.ClassNameIs (Name As String)
```

**Delphi Syntax**

```delphi
(Boolean :=) TNBAidHook.ClassNameIs (Name: string);
```
4.2.3.2.3.2 LoadFromXMLFile

Loads the XML code corresponding to the TNBAidHook object.

Visual Basic Syntax

TNBAidHook.LoadFromXMLFile (XMLFile As String)

Delphi Syntax

TNBAidHook.LoadFromXMLFile (XMLFile: string);

See Also SaveToXMLFile method.

4.2.3.2.3.3 SaveToXMLFile

Exports the XML code corresponding to the TNBAidHook object.

Visual Basic Syntax

TNBAidHook.SaveToXMLFile (XMLFile As String)

Delphi Syntax

TNBAidHook.SaveToXMLFile (XMLFile: string);

See Also LoadFromXMLFile method.

4.2.3.2.3.4 ShowEditor

Shows a modal dialog with the screen properties.

Visual Basic Syntax

TNBAidHook.ShowEditor

Delphi Syntax

TNBAidHook.ShowEditor;
4.2.3.2.4 Events

4.2.3.2.4.1 OnAidKey

Occurs when a combination of keys in the mapping table is pressed.

**Visual Basic Syntax**

TNBAidHook_OnAidKey (ByVal Value As String)

**Delphi Syntax**

TNBAidHook.OnAidKey (Sender: TObject);

**Remarks**

Value parameter can be one of the following strings:

**Local processing function keys:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up</td>
<td>go to previous row</td>
</tr>
<tr>
<td>Down</td>
<td>go to next row</td>
</tr>
<tr>
<td>Left</td>
<td>go to previous column</td>
</tr>
<tr>
<td>Right</td>
<td>go to next column</td>
</tr>
<tr>
<td>Insert</td>
<td>set the keyboard in inserting state</td>
</tr>
<tr>
<td>Delete</td>
<td>delete the next character</td>
</tr>
<tr>
<td>BackSpace</td>
<td>delete the previous character</td>
</tr>
<tr>
<td>ScreenBegin</td>
<td>go to row 1 column 1</td>
</tr>
<tr>
<td>ScreenEnd</td>
<td>go to last row last column</td>
</tr>
<tr>
<td>NextLine</td>
<td>go to column 1 of the next line</td>
</tr>
<tr>
<td>PriorField</td>
<td>go to the previous field</td>
</tr>
<tr>
<td>NextField</td>
<td>go to the next field</td>
</tr>
<tr>
<td>Home</td>
<td>go to the first field</td>
</tr>
<tr>
<td>End</td>
<td>go to last character of current field</td>
</tr>
<tr>
<td>EraseField</td>
<td>erase the current field</td>
</tr>
<tr>
<td>EraseEof</td>
<td>erase characters from current position until the end of field</td>
</tr>
<tr>
<td>FieldMinus</td>
<td>skip to the previous field</td>
</tr>
<tr>
<td>FieldPlus</td>
<td>skip to the next field</td>
</tr>
<tr>
<td>Restore</td>
<td>restore the current screen</td>
</tr>
</tbody>
</table>

**Host processing function keys (Attention Identifier "AID" keys):**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
</table>
Enter | ENTER key
---|---
Clear | CLEAR key
ATTN | Attention key
SysReq | System Request key
PF1 | PF01 key
PF2 | PF02 key
PF3 | PF03 key
PF4 | PF04 key
PF5 | PF05 key
PF6 | PF06 key
PF7 | PF07 key
PF8 | PF08 key
PF9 | PF09 key
PF10 | PF10 key
PF11 | PF11 key
PF12 | PF12 key
PF13 | PF13 key
PF14 | PF14 key
PF15 | PF15 key
PF16 | PF16 key
PF17 | PF17 key
PF18 | PF18 key
PF19 | PF19 key
PF20 | PF20 key
PF21 | PF21 key
PF22 | PF22 key
PF23 | PF23 key
PF24 | PF24 key
PA1 | PA1 key

See Also AIDKey property, AidKeyPressed, LocalKeyPressed, SendKeys and SendAid method.

4.2.3.2.5 Constants

4.2.3.2.5.1 TNBKbdKind

These values are used in the **KbdKind property** of telnet components.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>k5250</td>
<td>5250 Keyboard</td>
</tr>
<tr>
<td>k3270</td>
<td>3270 Keyboard</td>
</tr>
</tbody>
</table>
4.2.3.3 TTnbStatusBar Component

4.2.3.3.1 TTnbStatusBar Component

This component implements a status bar that shows the connection and session status.

**Properties**

- **Font**
- **TnbCom**
- **TnbEmulator**
- **Visible**

**Methods**

- **About**

4.2.3.3.2 Properties

4.2.3.3.2.1 Font

Sets/gets the font for the current connection.

**Visual Basic Syntax**

\[
\text{TNBStatusBar.Font} \; [= \; \text{IFontDisp}]
\]

**Delphi Syntax**

\[
\text{TNBStatusBar.Font} \; [= \; \text{IFontDisp}];
\]

4.2.3.3.2.2 TnbCom

Sets the TNB3270 or TNB5250 component as the telnet client component.

**Visual Basic Syntax**

\[
\text{TNBStatusBar.TnbCom} \; [= \; \text{TNBnnnn}]
\]

**Delphi Syntax**

\[
\text{TNBStatusBar.TnbCom} \; [= \; \text{TNBnnnn}];
\]
Remarks
You must set this property for TNBStatusBar component works properly.

4.2.3.3.2.3 TnbEmulator
Sets the TNBEmulator component for emulator events handling.

**Visual Basic Syntax**

```
TNBStatusBar.TnbEmulator [= TNBEmulator]
```

**Delphi Syntax**

```
TNBStatusBar.TnbEmulator [= TNBEmulator];
```

Remarks
You must set this property for TNBStatusBar component works properly.

4.2.3.3.2.4 Visible
Sets/gets the visibility of the control at run-time.

**Visual Basic Syntax**

```
TNBStatusBar.Visible [= Boolean]
```

**Delphi Syntax**

```
TNBStatusBar.Visible [= Boolean];
```

Remarks
Default value is True.
4.2.3.3.3 Methods

4.2.3.3.3.1 About

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

`TNBStatusBar.About`

**Delphi Syntax**

`TNBStatusBar.About;`

4.2.3.4 TTnbMacro Component

4.2.3.4.1 TTnbMacro Component

This component implements a way to record and play send/receive screen sequences. In a sequence you have fields list, cursor positions, Aid keys, etc.

**Properties**

- `AutoStartMacro`
- `MacroName`
- `State`
- `SubKey`
- `TnbCom`
- `TnbProfiles`

**Methods**

- `About`
- `LoadFromXMLFile`
- `SaveToXMLFile`
- `StartPlaying`
- `StartRecording`
- `StopPlaying`
- `StopRecording`

**Events**

- `OnStartPlaying`
- `OnStartRecording`
- `OnStopPlaying`
- `OnStopRecording`
4.2.3.4.2 Properties

4.2.3.4.2.1 AutoStartMacro

Sets/gets the macro name that will be auto started when the connection has been successfully established.

**Visual Basic Syntax**

```
TNBMacro.AutoStartMacro [= String]
```

**Delphi Syntax**

```
TNBMacro.AutoStartMacro [:= String];
```

**Remarks**

If you have assigned the TnbProfiles property to a TNBProfiles component, this property refers to a macro stored in the file pointed by this control; otherwise you must specify a valid filename of a macro file previously saved.

**See Also** TnbProfiles property, StopRecording and StartPlaying methods.

4.2.3.4.2.2 MacroName

Returns the name of the macro loaded. Sets the macro to be loaded from the TnbProfiles or a file. You can have one file for each macro, or one file containing several macros.

**Visual Basic Syntax**

```
TNBMacro.MacroName [= String]
```

**Delphi Syntax**

```
TNBMacro.MacroName [:= String];
```

**Remarks**

If you have assigned the TnbProfiles property to a TNBProfiles component, you can retrieve a specific macro by its name using this property. Otherwise, the MacroName property must specify a valid filename containing a macro previously saved.

**See Also** TnbProfiles property, StopRecording and StartPlaying methods.
4.2.3.4.2.3 State

Returns the state of the component.

**Visual Basic Syntax**

\[ \text{TNBMacroState} = \text{TNBMacro}\text{.State} \]

**Delphi Syntax**

\[ \text{TNBMacroState} := \text{TNBMacro}\text{.State}; \]

**Remarks**

The value of this property can be one of the following constants:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>msNoMacro</td>
<td>No macro is loaded.</td>
</tr>
<tr>
<td>msStopped</td>
<td>The component has a macro loaded and is in a stopped state.</td>
</tr>
<tr>
<td>msRecording</td>
<td>The component is recording a macro.</td>
</tr>
<tr>
<td>msPlaying</td>
<td>The component is playing the macro loaded.</td>
</tr>
</tbody>
</table>

See Also `TnbProfiles` property, `StopRecording` and `StartPlaying` methods.

4.2.3.4.2.4 SubKey

Sets/gets the subkey under which will be stored the macro sequences.

**Visual Basic Syntax**

\[ TNBMacro\text{.SubKey} [= \text{String}] \]

**Delphi Syntax**

\[ TNBMacro\text{.SubKey} [:= \text{String}]; \]

**Remarks**
This property is valid only if a TnbProfiles property has been assigned to a TNBProfiles component.

See also TnbProfiles property and TNBProfiles component.

4.2.3.4.2.5 TnbCom

Sets the TNB3270 or TNB5250 component as the telnet client component.

**Visual Basic Syntax**

```
TNBMacro.TnbCom [= TNBnnnn]
```

**Delphi Syntax**

```
TNBMacro.TnbCom [= TNBnnnn];
```

**Remarks**

You must set this property for TNBMacro component works properly.

4.2.3.4.2.6 TnbProfiles

Sets the TNBProfiles component as the profile manager for this control.

**Visual Basic Syntax**

```
TNBMacro.TnbProfiles [= TNBProfiles]
```

**Delphi Syntax**

```
TNBMacro.TnbProfiles [:= TNBProfiles];
```

**Remarks**

After setting this property, you can save and load macros from the file pointed in TNBProfiles component. You can retrieve a specific macro by its name using the MacroName property.

See Also TNBProfiles component, SubKey and MacroName properties.
4.2.3.4.3 Methods

4.2.3.4.3.1 About

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

`TNBMacro.About`

**Delphi Syntax**

`TNBMacro.About;`

4.2.3.4.3.2 LoadFromXMLFile

Loads the XML code corresponding to the `TNBMacro` object.

**Visual Basic Syntax**

`TNBMacro.LoadFromXMLFile (FileName As TFileName)`

**Delphi Syntax**

`TNBMacro.LoadFromXMLFile (FileName: TFileName);`

4.2.3.4.3.3 SaveToXMLFile

Exports the XML code corresponding to the `TNBMacro` object.

**Visual Basic Syntax**

`TNBMacro.SaveToXMLFile (FileName As TFileName)`

**Delphi Syntax**

`TNBMacro.SaveToXMLFile (FileName: TFileName);`
4.2.3.4.3.4 StartPlaying

Starts playing a macro.

**Visual Basic Syntax**

```vb
TNBMacro.StartPlaying (Name as String)
```

**Delphi Syntax**

```delphi
TNBMacro.StartPlaying (Name:string);
```

**Remarks**

Starts playing the macro specified in the Name parameter. The Name parameter follow the same rules as `MacroName` property.

See also `MacroName` property, `StopPlaying` method and `OnStartPlaying` event.

4.2.3.4.3.5 StartRecording

Starts the macro recording process.

**Visual Basic Syntax**

```vb
TNBMacro.StartRecording()
```

**Delphi Syntax**

```delphi
TNBMacro.StartRecording();
```

**Remarks**

Starts recording send and receive sequences. The process stops with the `StopRecording` method.

See also `StopRecording` method and `OnStartRecording` event.
4.2.3.4.3.6 StopPlaying

Stops playing the macro.

**Visual Basic Syntax**

TNBMacro.StopPlaying()

**Delphi Syntax**

TNBMacro.StopPlaying();

**Remarks**

Stops playing the current macro. Normally the macro stops when all its steps were successfully executed. However, in some cases the macro doesn't stop automatically because of changes on the screen sequences or field positions.

See also **StartPlaying** method **OnStopPlaying** event.

4.2.3.4.3.7 StopRecording

Stops the macro recording process.

**Visual Basic Syntax**

TNBMacro.StopRecording (Name as String)

**Delphi Syntax**

TNBMacro.StopRecording (Name:string);

**Remarks**

Stops the macro recording process and save the macro in a file. The Name parameter follow the same rules of the **MacroName** property.

See also **MacroName** property, **StartRecording**, **StartPlaying** methods and **OnStopRecording** event.
4.2.3.4.4 Events

4.2.3.4.4.1 OnStartPlaying
The event is fired when the playing process starts.

See also StartPlaying method.

4.2.3.4.4.2 OnStartRecording
The event is fired when the recording process starts.

See also StartRecording method.

4.2.3.4.4.3 OnStopPlaying
The event is fired when the playing process stops.

See also StopPlaying method.

4.2.3.4.4.4 OnStopRecording
The event is fired when the recording process stops.

See also StopRecording method.

4.2.3.4.5 Constants

4.2.3.4.5.1 TNBMacroState
These constants are used by the State property and let you know the current state of the component.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>msNoMacro</td>
<td>No macro is loaded.</td>
</tr>
<tr>
<td>msStopped</td>
<td>The component has a macro loaded and</td>
</tr>
</tbody>
</table>
4.2.3.5 TTnbProfiles Component

4.2.3.5.1 TTnbProfiles Component

This component implements a way to store profiles in an OLE Compound Document file or an XML file.

**Properties**

- **FileName**
- **Items**
- **Key**
- **ReadOnly**
- **StreamingType**

**Methods**

- **About**
- **Commit**
- **Delete**
- **GetCurrent**
- **GetXMLHeader**
- **Rename**
- **Rollback**
- **SetCurrent**

4.2.3.5.2 Properties

4.2.3.5.2.1 FileName

Sets or gets the file name for a profile. This file will be used to save a profile or to load an existing one.

**Visual Basic Syntax**

TNBProfiles.FileName [= String]

**Delphi Syntax**

TNBProfiles.FileName [:= String];
Remarks

Internally a profile file is divided in sections and subsections (keys and subkeys). You can use only one file for all kind of profiles you want to save, for example: connections, keyboard maps, macros, screen styles and hotspots, or one file for each kind of profile.

Sections can be accessed through **Key** property and subsections through **SubKey** property.

If you don't specify a directory, the file must be located in the component's directory (the directory where the component is installed). We recommend to use the *tbp* extension to identify a profile file (for example: *hollis.tbp*).

**See Also** **Key** property, **SubKey** property of TNB5250/TNB3270 components, **SubKey** property of TNBEmulator control, **SubKey** property of TNBAidHook control, **SubKey** property of TNBMacro control and **SubKey** property TNBHotSpots collection.

4.2.3.5.2.2 Items

Returns a String list containing the profiles stored under the **Key** property and the **Subkey** parameter. The values of **Key** property and **Subkey** parameter are in the file name specified in **FileName** property.

**Visual Basic Syntax**

```vbnet
OleVariant = TNBProfiles.Items (SubKey As String)
```

**Delphi Syntax**

```delphi
OleVariant := TNBProfiles.Items [SubKey:string];
```

**Remarks**

You can use the **Items.Count** property to know the total number of items in the list.

**See Also** **FileName** and **Key** properties.

4.2.3.5.2.3 Key

Sets or gets the key that will be used to store the profile.

**Visual Basic Syntax**
TNBProfiles.Key [= String]

**Delphi Syntax**

`TNBProfiles.Key [:= String];`

**Remarks**

If you share the file between two or more TNBProfiles components, the Key property must be different in order to store the profiles in different paths.

*See Also* [FileName] property.

---

### 4.2.3.5.2.4 ReadOnly

Indicates if the profile file has read only attribute set.

**Visual Basic Syntax**

`[Boolean =] TNBProfiles.ReadOnly`

**Delphi Syntax**

`[Boolean :=] TNBProfiles.ReadOnly;`

**Remarks**

If the attribute means that the profile file is read-only (ReadOnly property returning *True* value), profile information will not be saved preventing and error message to occur.

*See Also* [FileName] property.

---

### 4.2.3.5.2.5 StreamingType

Sets/gets the profile's streaming type.

**Visual Basic Syntax**

`TNBProfiles.StreamingType [= TStreamingType]`
Delphi Syntax

TNBProfiles.StreamingType [:= TStreamingType];

Remarks

TStreamingType can be either XML or OleDoc format. This format will be used to read and write profiles.

4.2.3.5.3 Methods

4.2.3.5.3.1 About

Shows Integration Pack about dialog box.

Visual Basic Syntax

TNBProfiles.About

Delphi Syntax

TNBProfiles.About;

4.2.3.5.3.2 Commit

Writes the XML configuration to the profile.

Visual Basic Syntax

TNBProfiles.Commit

Delphi Syntax

TNBProfiles.Commit;

See also Rollback method.
4.2.3.5.3.3 Delete

Deletes the specified profile.

**Visual Basic Syntax**

```vbnet
TNBProfiles.Delete (SubKey As String, Profile As String)
```

**Delphi Syntax**

```delphi
TNBProfiles.Delete (SubKey, Profile:string);
```

**Remarks**

SubKey parameter is the subkey of the caller control and Profile parameter is the name of the configuration that is going to be deleted.

See also Key and FileName properties, and Rename method.

4.2.3.5.3.4 GetCurrent

Returns a string with the current profile.

**Visual Basic Syntax**

```vbnet
[String :=] TNBProfiles.GetCurrent (Source As String)
```

**Delphi Syntax**

```delphi
[String :=] TNBProfiles.GetCurrent (Source: string);
```

**Remarks**

This method is generally used to get the default profile name to be loaded.

4.2.3.5.3.5 GetXMLHeader

Gets the XML Header with the Name, Tag, Id and Type attributes.

**Visual Basic Syntax**

```vbnet
TNBProfiles.GetXMLHeader (Tag As String, Name As String, Id As String, Type_ As
```
4.2.3.5.3.6 Rename

Renames the specified profile.

**Visual Basic Syntax**

```vbnet
TNBProfiles.Rename (Source As String, OldProfile As String, Profile As String, Id As String= '', type_ As String = '')
```

**Delphi Syntax**

```delphi
TNBProfiles.Rename (Source:string;OldProfile, Profile: string; Id:string= ''; type_: string = '');
```

**Remarks**

OldProfile parameter is the name of the profile that is going to be renamed and Profile parameter is the new name of the profile.

See also **Key** and **FileName** properties, and **Delete** method.

4.2.3.5.3.7 Rollback

Rollbacks changes done to the XML configuration.

**Visual Basic Syntax**

```vbnet
TNBProfiles.Rollback
```

**Delphi Syntax**
4.2.3.5.3.8 SetCurrent

Sets the current profile.

**Visual Basic Syntax**

```
TNBProfiles.SetCurrent (Source As String, CurrentProfile As String, id As String=' ',
type_ As String=' ')
```

**Delphi Syntax**

```
TNBProfiles.SetCurrent (Source:string;CurrentProfile: string;id:string=' ';type_:
string=' ');
```

**Remarks**

This method is generally used to set the default profile name to be loaded the next time the application starts.

4.2.3.6 TTnbHotSpots Class Reference

4.2.3.6.1 TTnbHotSpots Class Reference

HotSpots allows you transform screen text associated with terminal emulation tasks, to active elements such as buttons, links, etc. These active elements are defined through a pattern (sample text or regular expression) and will execute keystroke sequences through a simple mouse click. Also HotSpots are delimited to a specific screen area by row-col coordinates. The primary objective of a hotspot is to replace aid keys in a graphic form, and are direct actions of commands described in the host screen.

**Properties**

- `ActionFieldData`
- `Active`
- `Background`
- `Color`
- `Description`
### Constants

- **TTnbHotSpotViewAs**

---

**4.2.3.6.2 Properties**

**4.2.3.6.2.1 ActionFieldData**

Sets or gets the data to be sent to internal SendKeys method. Here you can specify key sequences to the mainframe, including AID keys. The syntax to be used for the data to be sent to the mainframe, is the same that uses SendKeys method of TNB5250/TNB3270/TNBSync components.

**Visual Basic Syntax**

```
TNBHotSpot.ActionFieldData [= String]
```

**Delphi Syntax**

```
TNBHotSpot.ActionFieldData [:= String];
```

**Remarks**

For a complete guide of how to complete the string to be assigned to this property, see table lists of functions keys and its corresponding codes in SendKeys method of TNB5250/TNB3270 components or SendKeys method of TNBSync component.

A simple example:

```
TNBHotSpot1.ActionFieldData:"L@E"
```

**See Also** [SendKeys](#) method (TNB5250/TNB3270), [SendKeys](#) method (TNBSync).
4.2.3.6.2.2 Active

Enables/disables the hotspot.

**Visual Basic Syntax**

```vbnet
TNBHotSpot.Active [= Boolean]
```

**Delphi Syntax**

```delphi
TNBHotSpot.Active [:= Boolean];
```

**Remarks**

In order to function properly, hotspot component must be active and also link with `TnbEmulator` component.

Default value for this property is **True**.

**See Also** `TnbEmulator` component, `HotSpotControl` property.

4.2.3.6.2.3 Background

Sets the hotspot background color to be shown in a host screen.

**Visual Basic Syntax**

```vbnet
TNBHotSpot.Background [= OLE_COLOR]
```

**Delphi Syntax**

```delphi
TNBHotSpot.Background [:= OLE_COLOR];
```

**Remarks**

This property returns an integer value that represent a hotspot background color.

**See Also** `Color` and `ViewAs` properties.
4.2.3.6.2.4 Color

Sets the hotspot foreground color to be shown in a host screen.

**Visual Basic Syntax**

```
TNNHotSpot.Color [= TColor]
```

**Delphi Syntax**

```
TNNHotSpot.Color [= TColor];
```

**Remarks**

This property is an integer value that represent a hotspot foreground color.

*See Also* Background and ViewAs properties.

4.2.3.6.2.5 Description

Sets/gets the description of the hotspot. This is simply a descriptive string associated with a hotspot.

**Visual Basic Syntax**

```
TNNHotSpot.Description [= String]
```

**Delphi Syntax**

```
TNNHotSpot.Description [:= String];
```

*See Also* Id property.

4.2.3.6.2.6 EndCol

Sets/gets the column where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.
Visual Basic Syntax

TNBHotSpot.EndCol [= Integer]

Delphi Syntax

TNBHotSpot.EndCol [= Integer];

Remarks

Valid values are from 1 to TNB5250/TNB3270.Cols property.

See Also EndRow, StartCol, StartRow properties.

4.2.3.6.2.7 EndRow

Sets/gets the row where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

Visual Basic Syntax

TNBHotSpot.EndRow [= Integer]

Delphi Syntax

TNBHotSpot.EndRow [= Integer];

Remarks

Valid values are from 1 to TNB5250/TNB3270.Rows property.

See Also StartRow, EndCol, StartCol properties.

4.2.3.6.2.8 Id

Unique Id of the hotspot. This is only for internal use.

Visual Basic Syntax
TNBHotSpot.Id [= String]

**Delphi Syntax**

TNBHotSpot.Id [= String];

See Also **Description** property.

### 4.2.3.6.2.9 Pattern

Regular host screen search expression for hotspot recognition and its graphical draw. When a hotspot is set in your application design time, it needs a search expression where Integration Pack can identify and display the hotspot, at application run time.

**Visual Basic Syntax**

TNBHotSpot.Pattern [= String]

**Delphi Syntax**

TNBHotSpot.Pattern [= string];

**Remarks**

This property is a string value that represent part or all the text displayed of a host screen field where you want to display a hotspot instead of the original text.

A simple example:

Suppose that a host screen has a field that denotes how you can access a specific application option, for example, accessing a public library with the command **L**. The text displayed in the host field will be like this:

**L - LIBRARY (Public Access)**

The corresponding value for pattern property will be:

TNBHotSpot1.Pattern:="L - LIBRARY \(Public Access\)"

In this case we use \ character because parenthesis characters in regular expression has a different meaning. If you don't have regular expressions characters in the host screen field, you can assign the same text that you are viewing (without \ characters).

Associated with Pattern property, ActionFieldData property must be set. In this example the value of this second property will be:
TNBHotSpot1.ActionFieldData:="L@E"

See Also ActionFieldData property.

4.2.3.6.2.10 StartCol

Sets/gets the column where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

**Visual Basic Syntax**

```vbnet
TNBHotSpot.StartCol [= Integer]
```

**Delphi Syntax**

```delphi
TNBHotSpot.StartCol [: = Integer];
```

**Remarks**

Valid values are from 1 to TNB5250/TNB3270.Cols property.

See Also EndRow, EndCol, StartRow properties.

4.2.3.6.2.11 StartRow

Sets/gets the row where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates. The coordinates must be set with correct row/col values in order to identify and display the corresponding hotspot.

**Visual Basic Syntax**

```vbnet
TNBHotSpot.StartRow [= Integer]
```

**Delphi Syntax**

```delphi
TNBHotSpot.StartRow [: = Integer];
```
Remarks

Valid values are from 1 to \textit{TNB5250/TNB3270.Rows} property.

See Also \texttt{EndRow}, \texttt{EndCol}, \texttt{StartCol} properties.

4.2.3.6.2.12 TextColor

Sets the hotspot foreground color to be shown in a host screen.

\textbf{Visual Basic Syntax}

\begin{verbatim}
TNBHotSpot.TextColor [= OLE_COLOR]
\end{verbatim}

\textbf{Delphi Syntax}

\begin{verbatim}
TNBHotSpot.TextColor [:= OLE_COLOR];
\end{verbatim}

Remarks

This property is a long value that represent a hotspot foreground color.

See Also \texttt{Background} and \texttt{ViewAs} properties.

4.2.3.6.2.13 ViewAs

Sets/gets the display format of the hotspot. A hotspot can be displayed in four formats: plain text, link, button or none.

\textbf{Visual Basic Syntax}

\begin{verbatim}
TNBHotSpot.ViewAs [= TTnbHotSpotViewAs]
\end{verbatim}

\textbf{Delphi Syntax}

\begin{verbatim}
TNBHotSpot.ViewAs [:= TTnbHotSpotViewAs];
\end{verbatim}

It can take one of this constants values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsButton</td>
<td>Hotspot button style is displayed.</td>
</tr>
<tr>
<td>hsLink</td>
<td>Hotspot link style is displayed.</td>
</tr>
</tbody>
</table>
### 4.2.3.6.3 Constants

**4.2.3.6.3.1 TTnbHotSpotViewAs**

These values are used in the `ViewAs` property of hotspot component.

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsButton</td>
<td>Hotspot button style is displayed.</td>
</tr>
<tr>
<td>hsLink</td>
<td>Hotspot link style is displayed.</td>
</tr>
<tr>
<td>hsNone</td>
<td>No hotspot is displayed.</td>
</tr>
<tr>
<td>hsPlain</td>
<td>Hotspot plain text style is displayed.</td>
</tr>
</tbody>
</table>

---

### 4.2.3.7 TTnbHotSpots Component

**4.2.3.7.1 TNBHotSpots Component**

Represent a collection of `TNBHotSpot` components. HotSpots allows you to work with mainframe applications using the mouse.

**Properties**

- `EndCol`
- `EndRow`
- `HotSpotName`
- `Items`
- `StartCol`
- `StartRow`
- `Subkey`
- `TnbProfiles`
- `ViewAs`

**Methods**

- `About`
- `Add`
- `Delete`
- `LoadFromXMLFile`
- `SaveToXMLFile`
- `ShowEditor`
4.2.3.7.2 Properties

4.2.3.7.2.1 EndCol

Sets/gets the column default value where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

```
[Integer =] TNBHotSpots.EndCol
```

**Delphi Syntax**

```
[Integer :=] TNBHotSpots.EndCol;
```

**Remarks**

Return value is between 1 and TNB5250/TNB3270.Rows property.

See Also **EndRow, StartCol, StartRow** properties.

4.2.3.7.2.2 EndRow

Sets/gets the row default value where the hotspot recognition area of the host screen ends. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

```
[Integer :=] TNBHotSpots.EndRow
```

**Delphi Syntax**

```
[Integer =] TNBHotSpots.EndRow;
```

**Remarks**

Return value is between 1 and TNB5250/TNB3270.Rows property.

See Also **EndCol, StartCol, StartRow** properties.
4.2.3.7.2.3 HotSpotName

Sets/gets the description of the hotspot's default value. This is simply a descriptive string that will be associated with a hotspot at first.

**Visual Basic Syntax**

`TNBHotSpot.HotSpotName [= String]`

**Delphi Syntax**

`TNBHotSpot.HotSpotName [= String];`

**See Also** `Description`, `Id` properties.

4.2.3.7.2.4 Items

Contains a collection of `TNBHotSpot` objects.

**Visual Basic Syntax**

`TNBHotSpots.Items (Index As Integer) [= TNBHotSpot]`

**Delphi Syntax**

`TNBHotSpots.Items [Index:integer] [= TNBHotSpot];`

**Remarks**

Use Items to get an specific hotspot from the array. The Index parameter indicates the object's index in the collection, where 0 is the index of the first element and (TNBHotSpots.Items.Count-1) is the index of the last element. Use Items with the Count property to iterate through all the objects in the list.

**See also** `TNBHotSpot` component.
4.2.3.7.2.5 StartCol

Sets/gets the column default value where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

\[ \text{Integer} = \text{TNBHotSpots.StartCol} \]

**Delphi Syntax**

\[ \text{Integer} := \text{TNBHotSpots.StartCol}; \]

**Remarks**

Return value is between 1 and \text{TNB5250/TNB3270.Cols} property.

**See Also** \text{EndRow}, \text{EndCol}, \text{StartRow} properties.

4.2.3.7.2.6 StartRow

Sets/gets the row default value where the hotspot recognition area of the host screen begins. Integration Pack searches for a hotspot into a host screen area delimited for: start row, start column, end row and end column coordinates.

**Visual Basic Syntax**

\[ \text{Integer} = \text{TNBHotSpots.StartRow} \]

**Delphi Syntax**

\[ \text{Integer} := \text{TNBHotSpots.StartRow}; \]

**Remarks**

Return value is between 1 and \text{TNB5250/TNB3270.Rows} property.

**See Also** \text{EndCol}, \text{StartCol}, \text{EndRow} properties.
4.2.3.7.2.7 Subkey

Sets/gets the subkey under which will be stored the hotspot profile.

**Visual Basic Syntax**

```
TNBHotSpots.SubKey [= String]
```

**Delphi Syntax**

```
TNBHotSpots.SubKey := String;
```

**Remarks**

This property is valid only if a \texttt{TnbProfiles} property has been assigned to a \texttt{TNBProfiles} component.

See also \texttt{TnbProfiles} property and \texttt{TNBProfiles} component.

4.2.3.7.2.8 TnbProfiles

Sets the \texttt{TNBProfiles} component as the profile manager for this control.

**Visual Basic Syntax**

```
TNBHotSpots.TnbProfiles [= TNBProfiles]
```

**Delphi Syntax**

```
TNBHotSpots.TnbProfiles := TNBProfiles;
```

See Also \texttt{TNBProfiles} component and \texttt{ShowEditor} method.

4.2.3.7.2.9 ViewAs

Gets/sets the display format default value of a hotspot. A hotspot can be displayed in four formats: plain text, link, button or none.

**Visual Basic Syntax**

```
[TTnbHotSpotViewAs =] TNBHotSpot.ViewAs
```

© 2015, Cybele Software, Inc.
### Delphi Syntax

```delphi
[TTnbHotSpotViewAs :=] TNBHotSpot.ViewAs;
```

It can take one of this constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsButton</td>
<td>Hotspot button style is displayed.</td>
</tr>
<tr>
<td>hsLink</td>
<td>Hotspot link style is displayed.</td>
</tr>
<tr>
<td>hsNone</td>
<td>No hotspot is displayed.</td>
</tr>
<tr>
<td>hsPlain</td>
<td>Hotspot plain text style is displayed.</td>
</tr>
</tbody>
</table>

### 4.2.3.7.3 Methods

#### 4.2.3.7.3.1 About

Shows Integration Pack about dialog box.

**Visual Basic Syntax**

```vbnet
TNBHotSpots.About
```

**Delphi Syntax**

```delphi
TNBHotSpots.About;
```

#### 4.2.3.7.3.2 Add

 Adds a new TnbXHotspot to the **TNBHotSpots** collection.

**Visual Basic Syntax**

```vbnet
[TnbHotSpot =] TNBXHotSpots.Add
```

**Delphi Syntax**

```delphi
[TnbHotSpot :=] TNBXHotSpots.Add;
```

**See Also** **Delete** method.
4.2.3.7.3.3 Delete

Deletes a specified HotSpot of the TNBHotSpots collection.

**Visual Basic Syntax**

TNBHotSpots.Delete (Index As Integer)

**Delphi Syntax**

TNBHotSpots.Delete (Index: Integer);

See Also Add method.

4.2.3.7.3.4 LoadFromFile

Loads the XML code corresponding to the TNBHotSpots object.

**Visual Basic Syntax**

TNBHotSpots.LoadFromXMLFile (XMLFile As String)

**Delphi Syntax**

TNBHotSpots.LoadFromXMLFile (XMLFile: string);

4.2.3.7.3.5 SaveToFile

Exports the XML code corresponding to the TNBHotSpots object.

**Visual Basic Syntax**

TNBHotSpots.SaveToXMLFile (XMLFile As String)

**Delphi Syntax**

TNBHotSpots.SaveToXMLFile (XMLFile: string);
4.2.3.7.3.6 ShowEditor

Shows a modal dialog with the *TNBHotSpot*’s properties.

**Visual Basic Syntax**

*TNBHotSpots*.ShowEditor

**Delphi Syntax**

*TNBHotSpots*.ShowEditor;

See also *TnbProfiles* property.

4.2.4 Helper Controls

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack Delphi Helper Controls.

4.2.4.1 TTnbSync Component

4.2.4.1.1 TTnbSync Component

This component gives a simplified interface for programming synchronously with TNB3270 and TNB5250 components (in a synchronous mode).

**Properties**

- EmptyScreenDelay
- TnbCom
- TnbTrace
- WaitTimeout

**Methods**

- Connect
- Disconnect
- IsConnected
- SendKeys
- Wait
- WaitFor
- WaitForConnect
- WaitForData
- WaitForDisconnect
- WaitForNewScreen
• **WaitForNewUnlock**
• **WaitForScreen**
• **WaitForUnlock**

4.2.4.1.2 Properties

4.2.4.1.2.1 EmptyScreenDelay

Sets/gets the additional wait time to bypass empty host screens. This time is measured in milliseconds.

**Visual Basic Syntax**

```vbnet
TNBSync.EmptyScreenDelay [= Integer]
```

**Delphi Syntax**

```delphi
TNBSync.EmptyScreenDelay [= Integer];
```

**Remarks**

When you make a call to the **Wait** method, if an empty screen arrives from the host and the system is in an unlocked state, an internal **WaitForNewScreen** method is made with a timeout taken from this property value. It allows to bypass intermediate empty screens without additional programming effort. It applies only to the **Wait** method and other methods that call a Wait internally like **Connect** and **SendKeys** methods.

**Default** value is 0.

See also **Connect**, **SendKeys** and **Wait** methods.

4.2.4.1.2.2 TnbCom

Sets/gets the TNB3270 or TNB5250 component as the telnet client control.

**Visual Basic Syntax**

```vbnet
TNBSync.TnbCom [= TNBnnnn]
```

**Delphi Syntax**

```delphi
TNBSync.TnbCom [= TNBnnnn];
```
Remarks
You must set this property for TNBSync component works properly.

4.2.4.1.2.3 TnbTrace
Sets the TNBTrace component as its trace agent.

Visual Basic Syntax
TNBSync.TnbTrace [= TNBTrace]

Delphi Syntax
TNBSync.TnbTrace := TNBTrace;

Remarks
You must set this property to get trace information from TNBSync component.

4.2.4.1.2.4 WaitTimeout
Sets/gets the default timeout value in milliseconds for all the wait methods.

Visual Basic Syntax
TNBSync.WaitTimeout [= Integer]

Delphi Syntax
TNBSync.WaitTimeout := Integer;

Remarks
You can assign any value, there isn't a limit of maximum value.

Default value is 60000 milliseconds (equivalent to 1 minute).

See also Wait, WaitFor, WaitForConnect, WaitForDisconnect,
WaitForNewScreen, WaitForNewUnlock, WaitForScreen and WaitForUnlock methods.

4.2.4.1.3 Methods

4.2.4.1.3.1 Connect

The Connect method establishes a synchronous connection to the Telnet server.

**Visual Basic Syntax**

\[\text{[Boolean]} := \text{TNBSync.Connect([TimeOut] As Integer)}\]

**Delphi Syntax**

\[\text{[Boolean]} := \text{TNBSync.Connect([TimeOut]:integer)};\]

The TimeOut parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the connection was successfully established and False if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

See also WaitForConnect and Wait methods.

4.2.4.1.3.2 Disconnect

The Disconnect method terminates the connection to the Telnet server.

**Visual Basic Syntax**

\[\text{[Boolean]} := \text{TNBSync.Disconnect([TimeOut] As Integer)}\]

**Delphi Syntax**

\[\text{[Boolean]} := \text{TNBSync.Disconnect([TimeOut]: integer)};\]

The TimeOut parameter is measured in milliseconds.
Remarks

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the disconnection was successfully established and False if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

See also WaitForDisconnect and Wait methods.

4.2.4.1.3.3 IsConnected

Returns a boolean value indicating if a connection with the host is currently established.

Visual Basic Syntax

[Boolean =] TNBSync.IsConnected

Delphi Syntax

[Boolean :=] TNBSync.IsConnected;

See also Connect and Disconnect methods.

4.2.4.1.3.4 SendKeys

The SendKeys method can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character ("@") and a mnemonic code that corresponds to the supported host functions.

Visual Basic Syntax

TNBSync.SendKeys (Keys As String)

Delphi Syntax

TNBSync.SendKeys (Keys: string);

Remarks
The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

This method complements the SendKeys method from TNB3270 and TNB5250, adding the possibility to send several key sequences separated by aid keys and waits (@W). This is the main difference between both SendKeys methods. This mean that you can navigate through several screens with a single method call.

See also TelnetControl property of TNBSync component and SendKeys method from TNB3270/TNB5250 components.
4.2.4.1.3.5 Wait

General wait mechanism. This method basically waits until the mainframe application turns in an unlocked state and is able to receive input data.

Visual Basic Syntax

[Boolean] = TNBSync.Wait([Timeout] As Integer)

Delphi Syntax

[Boolean] := TNBSync.Wait([Timeout]: integer);

The TimeOut parameter is measured in milliseconds.

Remarks

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the wait was successful and False if the timeout period has expired. This method uses alternatively the WaitForConnect, WaitForUnlock and WaitForNewScreen methods, according to the actual mainframe state (disconnected, locked or unlocked). If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

See also WaitTimeout property, WaitForConnect, WaitForUnlock and WaitForNewScreen methods.

4.2.4.1.3.6 WaitFor

Wait for an screen containing the specified string.

Visual Basic Syntax

[Boolean] := TNBSync.WaitFor(StrValue As String, [Timeout] As Integer)

Delphi Syntax

[Boolean] := TNBSync.WaitFor(StrValue: string; [Timeout]: integer);

The TimeOut parameter is measured in milliseconds.

Remarks
The Timeout parameter is optional, assuming the **WaitTimeout** property value by default. This method returns **True** if the wait was successful and **False** if the timeout period has expired.
If you pass a timeout value in the Timeout parameter of this method, the value of the property **WaitTimeout** will not have any effect.

**See also WaitTimeout** property, **WaitForScreen**, **WaitForUnlock** and **WaitForNewScreen** methods.

### 4.2.4.1.3.7 WaitForConnect

This method waits until the telnet connection is successfully opened, or the timeout period expires.

**Visual Basic Syntax**

```
[Boolean] = TNBSync.WaitForConnect([TimeOut] As Integer)
```

**Delphi Syntax**

```
[Boolean] := TNBSync.WaitForConnect([TimeOut]: integer);
```

The **TimeOut** parameter is measured in milliseconds.

**Remarks**

The Timeout parameter is optional, assuming the **WaitTimeout** property value by default. This method returns **True** if the wait was successful and **False** if the timeout period has expired.
If you pass a timeout value in the Timeout parameter of this method, the value of the property **WaitTimeout** will not have any effect.

**See also WaitTimeout** property and **Wait** method.

### 4.2.4.1.3.8 WaitForEndDoc

Waits until an end document event from host printed documents occurs. This method is used with **Tnb3287** and **Tnb3812** printer emulation controls and indicates the end of a document that was already printed.

**Visual Basic Syntax**
[Boolean] = TNBSync.WaitForEndDoc ([TimeOut] As Integer)

Delphi Syntax

[Boolean] := TNBSync.WaitForEndDoc ([TimeOut]: integer);

The TimeOut parameter is measured in milliseconds.

Remarks

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the wait was successful and False if the timeout period has expired. If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

See also WaitTimeout property and WaitForStartDoc method.

4.2.4.1.3.9 WaitForData

Waits until a new data screen appears. Internally makes a WaitForNewScreen.

Visual Basic Syntax

[Boolean] = TNBSync.WaitForData ([TimeOut] As Integer)

Delphi Syntax

[Boolean] := TNBSync.WaitForData ([TimeOut]: integer);

The TimeOut parameter is measured in milliseconds.

Remarks

The Timeout parameter is optional, assuming the WaitTimeout property value by default. This method returns True if the wait was successful and False if the timeout period has expired. If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

See also WaitTimeout property, WaitForScreen, WaitForUnlock and WaitForNewScreen methods.
4.2.4.1.3.10 WaitForDisconnect

This method waits until the telnet connection is successfully closed, or the timeout period expires.

**Visual Basic Syntax**

```vbnet
[Boolean] := TNBSync.WaitForDisconnect ([TimeOut] As Integer)
```

**Delphi Syntax**

```pascal
[Boolean] := TNBSync.WaitForDisconnect ([TimeOut]: integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

The `Timeout` parameter is optional, assuming the `WaitTimeout` property value by default. This method returns **True** if the wait was successful and **False** if the timeout period has expired.

If you pass a timeout value in the `Timeout` parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `Wait` method.

4.2.4.1.3.11 WaitForNewScreen

This method waits until a new screen arrives within the specified period of time.

**Visual Basic Syntax**

```vbnet
[Boolean] = TNBSync.WaitForNewScreen ([TimeOut] As Integer)
```

**Delphi Syntax**

```pascal
[Boolean] := TNBSync.WaitForNewScreen ([TimeOut]:integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

This method is similar to `WaitForScreen` method except that it always waits for a new screen arrival.

The `Timeout` parameter is optional, assuming the `WaitTimeout` property value by default. This method returns **True** if the wait was successful and **False** if the timeout
period has expired. If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `WaitForScreen`, `WaitForData` methods.

### 4.2.4.1.3.12 WaitForNewUnlock

This method waits until a new system unlock arrives within the specified period of time.

**Visual Basic Syntax**

```vbnet
[Boolean] = TNBSync.WaitForNewUnlock ([Timeout] As Integer)
```

**Delphi Syntax**

```delphi
[Boolean] := TNBSync.WaitForNewUnlock ([Timeout]:integer);
```

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

This method is similar to `WaitForUnlock` method except that it always waits for a new system unlock arrival. The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. This method returns `True` if the wait was successful and `False` if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `WaitForUnlock` method.

### 4.2.4.1.3.13 WaitForStartDoc

Waits until a start document event from the host, because of printed documents occurs. This method is used with `Tnb3287` and `Tnb3812` printer emulation controls and indicates the beginning of a document that was already printed.

**Visual Basic Syntax**

```vbnet
```

**Delphi Syntax**

```delphi
[Boolean] := TNBSync.WaitForStartDoc ([Timeout]:integer);
```


```delphi
[Boolean] := TNBSync.WaitForStartDoc ([Timeout]:integer);
```

The `Timeout` parameter is measured in milliseconds.

**Remarks**

The `Timeout` parameter is optional, assuming the `WaitTimeout` property value by default. This method returns `True` if the wait was successful and `False` if the timeout period has expired.

If you pass a timeout value in the `Timeout` parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `WaitForEndDoc` method.

---

### 4.2.4.1.3.14 WaitForScreen

This method waits for the first screen after a system lock.

**Visual Basic Syntax**

```vbnet
```

**Delphi Syntax**

```delphi
[Boolean] := TNBSync.WaitForScreen ([Timeout]: integer);
```

The `Timeout` parameter is measured in milliseconds.

**Remarks**

If the host system changes from an unlocked to a locked state (normally when we send an Aid key) the method waits until the first screen arrives. If `WaitForScreen` method is called after that event, it returns immediately.

The `Timeout` parameter is optional, assuming the `WaitTimeout` property value by default. The method returns `True` if the wait was successful and `False` if the timeout period has expired.

If you pass a timeout value in the `Timeout` parameter of this method, the value of the property `WaitTimeout` will not have any effect.

See also `WaitTimeout` property and `WaitForNewScreen` method.
4.2.4.1.3.15 WaitForUnlock

This method waits until the host system turns to an unlocked state.

**Visual Basic Syntax**

\[
\]

**Delphi Syntax**

\[
[Boolean] := TNBSync.WaitForUnlock([TimeOut]:integer);
\]

The `TimeOut` parameter is measured in milliseconds.

**Remarks**

If the host system is in an unlock state this method returns immediately. If the host system is in a lock state, the method waits until a system unlock event arrives (`OnSystemUnlock`).

The Timeout parameter is optional, assuming the `WaitTimeout` property value by default. The method returns **True** if the wait was successful and **False** if the timeout period has expired.

If you pass a timeout value in the Timeout parameter of this method, the value of the property `WaitTimeout` will not have any effect.

**See also** `WaitTimeout` property and `WaitForNewUnlock` method.

4.2.4.2 TTnbXmlTemplate Class

4.2.4.2.1 TTnbXmlTemplate Class

The `TTnbXmlTemplate` class implements the runtime side for working with XML Templates created with Development Lab. It implements the `IXmlProducer` interface and can be assigned to `Producer` property of `XmlBroker` to produce custom XML.

**Properties**

- **Id**

**Methods**

- **HostToXml**
- **XmlToHost**
- **CanProduce**

**Events**
4.2.4.2.2 Properties

4.2.4.2.2.1 Id

Gets an Id for the active XML Template.

**VB.NET Syntax**

```vbnet
[String =] XmlTemplate.Id
```

**C# Syntax**

```csharp
[String =] XmlTemplate.Id;
```

**Remarks**

Returns a string with an identification corresponding to the active XML Template. This Id is equivalent to the filename of the XML template excluding the extension.

4.2.4.2.3 Methods

4.2.4.2.3.1 CanProduce

Returns true if the XmlTemplate can produce Xml.

**VB.NET Syntax**

```vbnet
[Boolean =] XmlTemplate.CanProduce
```

**C# Syntax**

```csharp
[Boolean =] XmlTemplate.CanProduce;
```

**Remarks**

When you call CanProduce method it tries to find an XML template that matches with the current Screen. In case it finds it, returns true telling that it can produce an XML content. In other case returns false.
4.2.4.2.3.2 HostToXml

Returns a XML representation of the current mainframe screen.

**VB.NET Syntax**

```vbnet
[String =] XmlTemplate.HostToXml
```

**C# Syntax**

```csharp
[String =] XmlTemplate.HostToXml;
```

**Remarks**

Returns an XML content based on the XML template that matches with the current screen.

4.2.4.2.3.3 XmlToHost

Interpretes the input XML to fill the unprotected fields.

**VB.NET Syntax**

```vbnet
XmlTemplate.HostToXml(Xml As String)
```

**C# Syntax**

```csharp
XmlTemplate.HostToXml(Xml: String);
```

**Remarks**

Interpretes the XML passed as parameter and fill unprotected fields, based on the active XML template for the current screen.
4.2.4.2.4 Events

4.2.4.2.4.1 OnCustomXml

Occurs during the processing of a custom XML area.

Remarks

When this event fires you have the chance to produce your custom XML content for that area.

4.2.4.3 TTnbXmlBroker Component

4.2.4.3.1 TNBXmIBroker Component

This component converts all data received from the TnbCom component into XML code and when it receives XML code, sends it to the TnbCom component.

Properties

- ScreenName
- TnbCom
- XML

Methods

- LoadFromFile
- SaveToFile

See also XML Reference for TnbXmlBroker.

4.2.4.3.2 Properties

4.2.4.3.2.1 IncludeStatus

Determines whether the Status tag and inner elements are included or not. Default value is true.

Visual Basic Syntax

TNBXmIBroker.IncludeStatus [= Boolean]

Delphi Syntax
4.2.4.3.2.2 Producer

Assigns an XML object capable to produce XML content.

**Visual Basic Syntax**

```vbnet
TNBXmlBroker.Producer [= IXmlProducer]
```

**Delphi Syntax**

```delphi
TNBXmlBroker.Producer [:= IXmlProducer];
```

See also [TTnbXmlTemplate](#) class

4.2.4.3.2.3 ScreenName

Sets/gets the corresponding screen name.

**Visual Basic Syntax**

```vbnet
TNBXmlBroker.ScreenName [= String]
```

**Delphi Syntax**

```delphi
TNBXmlBroker.ScreenName [= String];
```

See also [XML Reference](#) for TnbXmlBroker.

4.2.4.3.2.4 TnbCom

Sets/gets the telnet client component for executing transactions with it.

**Visual Basic Syntax**

```vbnet
TNBXmlBroker.TnbCom [= TNBnnnn]
```
**Delphi Syntax**

\[ TNBXmlBroker.TnbCom := TNBnnnn; \]

### 4.2.4.3.2.5 XML

When assigning an XML code, it generates a `TnbFields` collection. Reading this property, generates and returns an XML code from unprotected `TnbFields`, cursor location and AID Key.

**Visual Basic Syntax**

\[ TNBXmlBroker.XML [= String] \]

**Delphi Syntax**

\[ TNBXmlBroker.XML [= String]; \]

See also [XML Reference](#) for TnbXmlBroker.

### 4.2.4.3.3 Methods

#### 4.2.4.3.3.1 LoadFromFile

Imports an XML file and then sets the `XML` property with it.

**Visual Basic Syntax**

\[ TNBXmlBroker.LoadFromFile (XMLFile As String) \]

**Delphi Syntax**

\[ TNBXmlBroker.LoadFromFile (XMLFile: string); \]

#### 4.2.4.3.3.2 SaveToFile

Exports the XML code corresponding to the `TNBXmlBroker` object with all the unprotected fields and the screen configuration data.
Visual Basic Syntax

TNBXmlBroker.SaveToFile (XMLFile As String)

Delphi Syntax

TNBXmlBroker.SaveToFile (XMLFile: string);

4.2.4.3.4 XML Reference

This XML code is generated automatically by the TnbXmlBroker control. The following is the tags' hierarchy it uses:

```xml
<TNBRIDGE>
  <status>
    <direction... />
    <transaction... />
    <cursor_pos... />
    <session... />
    <state... />
    <xmlscreen... />
    <cursor_field... />
    <timestamp... />
  </status>
  <screen>
    <rows... />
    <cols... />
    <type... />
    <model... />
    <fields>
      <field ...>
        <name... />
        <attr... />
        <value... />
      </field>
      <field />
    </fields>
  </screen>
</TNBRIDGE>
```
4.2.4.3.4.1 XML Tags and Attributes

This is the root tag and includes all the other tags. These are the following:

- **Status**: indicates connection characteristics.
- **Screen**: indicates screen's characteristics and contains a collection of screen fields.

See also **XML Example**.

Contains the following tags:

- **direction**: indicates INPUT or OUTPUT direction. When the XML is generated by the TnbXmlBroker control it is set with OUTPUT value.
- **transaction**: indicates transaction identifier.
- **cursor_pos**: indicates cursor location.
- **session**: LU-LU (in session with VTAM application), noSession (not in session) or SSCPLU (in session with VTAM).
- **state**: indicates LOCKED or UNLOCKED state.
- **xmlscreen**: indicates the screen name. You must set this property using the **ScreenName** property.
- **cursor_field**: indicates the field's name where the cursor is located.
- **timestamp**: indicates the date the file was created.

See also **XML Example** and **ScreenName** property.

Contains the following tags:

- **rows**: indicates the number of rows of the screen.
- **cols**: indicates the number of columns of the screen.
- **type**: indicates terminal type (3270 or 5250).
- **model**: indicates terminal model.
- **fields**: contains a collection of fields.

See also **XML Example**.

4.2.4.3.4.2 XML Example

Here you can see an example of XML Code:

```xml
<TNBRIDGE>
```
4.2.4.4 TTnbXmlClient Component

4.2.4.4.1 TNBXmlClient Component

This control acts like a virtual TNB3270/TNB5250 component. It generates the TnbFields necessary as input for the emulation control or any other user interface from XML code as well generates the XML code from the modified TnbFields, cursor location and 'pressed' AID key. When used in conjunction with the TNBXmIBroker component allows build a bridge between the 3270/5250 Telnet components and the emulation interface using XML streaming. This bridge could be implemented as a client/server communication using TCP/IP protocols, Web Services, etc.

Properties
• AidKey
• AidString
• ExcludeEmptyFields
• ScreenRow
• Text
• TnbHotSpots
• XML

Methods

• Connect
• Disconnect
• GetScreenRowEx
• GetScreenText
• IsConnected
• FindEditField
• LoadFromXMLFile
• SaveToXMLFile
• SendAid
• SendKeys
• SetConnected

Events

• OnConnect
• OnConnectionFail
• OnConnectRemote
• OnDisconnect
• OnDisconnectRemote
• OnScreenChange
• OnSendAid
• OnSystemLock
• OnSystemUnlock

4.2.4.4.2 Properties

4.2.4.4.2.1 AidKey

Sets the Attention Identifier Key and executes the SendAid method. Modified fields can be input fields (unprotected) or, sometimes protected fields, having the property Modified set to True.

Visual Basic Syntax

TNBXmlClient.AIDKey [= TNBAid]

Delphi Syntax
**TNXmIClient.AIDKey** ::= **TNBAid**;

It can take any of TNBAidKey constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>akPF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>akPF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>akPF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>akPF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>akPF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>akPF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>akPF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>akPF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>akPF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>akPF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>akPF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>akPF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>akPF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>akPF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>akPF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>akPF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>akPF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>akPA1</td>
<td>PA1 key</td>
</tr>
<tr>
<td>akReset</td>
<td>Reset key</td>
</tr>
<tr>
<td>akPgUp</td>
<td>Page Up key</td>
</tr>
<tr>
<td>akPgDown</td>
<td>Page Down key</td>
</tr>
<tr>
<td>akPgLeft</td>
<td>Page Left key</td>
</tr>
<tr>
<td>akPgRight</td>
<td>Page Right key</td>
</tr>
<tr>
<td>akPrint</td>
<td>Print key</td>
</tr>
<tr>
<td>akHelp</td>
<td>Help key</td>
</tr>
</tbody>
</table>

See also **TNBAid** constants.
4.2.4.4.2.2 AidString

Sets the corresponding Attention Identifier Key when a `SendAid` method is executed, and gets this value later if it is necessary.

**Visual Basic Syntax**

`TNBXmlClient.AIDString [= AidKey As String]`

**Delphi Syntax**

`TNBXmlClient.AIDString [= AidKey:String];`

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>PF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>PF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>PF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>PF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>PF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>PA1</td>
<td>PA1 key</td>
</tr>
</tbody>
</table>
See also SendAid method.

4.2.4.4.2.3 ExcludeEmptyFields
Sets/gets the possibility of excluding all empty fields.

**Visual Basic Syntax**

```vbnet
TNBXmIClient.ExcludeEmptyFields [= Boolean]
```

**Delphi Syntax**

```delphi
TNBXmIClient.ExcludeEmptyFields [= Boolean];
```

4.2.4.4.2.4 ScreenRow
Gets the text contained into the specified row.

**Visual Basic Syntax**

```vbnet
[String =] TNBXmIClient.ScreenRow (Index As Integer)
```

**Delphi Syntax**

```delphi
[String :=] TNBXmIClient.ScreenRow [Index:integer];
```

**Remarks**

Index parameter indicates the corresponding row number.

4.2.4.4.2.5 Text
Gets the text located in the column and row coordinates, with the length specified.

**Visual Basic Syntax**

```vbnet
[String =] TNBXmClient.Text (r As Integer, c As Integer, l As Integer)
```
**Delphi Syntax**

\[ String := TNBXmlClient.Text [r, c, l: Integer]; \]

**Remarks**

\( r \) and \( c \) indicate the row and column coordinates respectively and \( l \) the length of the string.

4.2.4.4.2.6 TnbHotSpots

Sets/gets the `TNBHotSpot` component as the hotspots handler for this control.

**Visual Basic Syntax**

\[ TNBXmlClient.TnbHotSpots [= TNBHotSpot] \]

**Delphi Syntax**

\[ TNBXmlClient.TnbHotSpots [= TNBHotSpot]; \]

**See Also** `TNBHotSpot` component.

4.2.4.4.2.7 XML

When assigning an XML code, it generates a `TnbFields` collection. Reading this property, generates and returns an XML code from unprotected `TnbFields`, cursor location and AID Key.

**Visual Basic Syntax**

\[ TNBXmlClient.XML [= String] \]

**Delphi Syntax**

\[ TNBXmlClient.XML [= String]; \]
4.2.4.4.3 Methods

4.2.4.4.3.1 Connect

The Connect method initializes the connection sequence process. It actually doesn't produce by itself a connection to a resource, but fires the `OnConnectRemote` event in order to give the change to the client program to establish the connection to the resource which will be the XML producer (a local or remote file, a local or remote `TnbXMLBroker`, a Web Service, etc).

**Visual Basic Syntax**

```vbnet
TNBXmlClient.Connect
```

**Delphi Syntax**

```delphi
TNBXmlClient.Connect;
```

See also `Disconnect` method, `OnConnectRemote`, `OnConnect` and `OnConnectionFail` events.

4.2.4.4.3.2 Disconnect

The Disconnect method ends the connection sequence process. It actually doesn't produce by itself a disconnection to a resource, but fires the `OnDisconnectRemote` event in order to give the change to the client program to end the connection to the resource.

**Visual Basic Syntax**

```vbnet
TNBXmlClient.Disconnect
```

**Delphi Syntax**

```delphi
TNBXmlClient.Disconnect;
```

See also `Connect` method, `OnDisconnectRemote` and `OnDisconnect` event.

4.2.4.4.3.3 GetScreenRowEx

GetScreenRowEx can be used to retrieve the text buffer of one row of the current screen. Additionally, you can retrieve the standard or extended attribute corresponding to each
field.

**Visual Basic Syntax**

```vbnet
[String =] TNBXmlClient.GetScreenRowEx (Row As Integer, [Attr As Boolean = True], [Eab As Boolean = True])
```

**Delphi Syntax**

```delphi
[String :=] TNBXmlClient.GetScreenRowEx (Row:integer, [Attr:boolean = True], [Eab:boolean = True])
```

**Remarks**

The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned string, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in an 80x24 display, each row returned will occupy 160 characters.

See also [GetScreenText](#) method.

---

### 4.2.4.4.3.4 GetScreenText

GetScreenText returns the text buffer of a portion of the current screen. You can choose to retrieve the attributes or the extended attributes of each field joined to the character data.

**Visual Basic Syntax**

```vbnet
[String =] TNBXmlClient.GetScreenText (StartPos As Integer, EndPos As Integer, [Attr As Boolean = True], [Eab As Boolean = True])
```

**Delphi Syntax**

```delphi
[String :=] TNBXmlClient.GetScreenText (StartPos:integer, EndPos:integer, [Attr:boolean = True], [Eab:boolean = True])
```

**Remarks**

StartPos and EndPos specify the start position and the end position of the text buffer.
that will be returned. The Attr parameter indicates that GetScreenRowEx will include the field attribute that occurs in the row specified. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format. The Eab parameter indicates that GetScreenRowEx will include the extended attribute that occurs in Row. The Eab is associated with individual characters. Each character in the buffer (except for field attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format. So, in a 80x24 display, each row returned will occupy 160 characters.

See also GetScreenRowEx method.

4.2.4.4.3.5 IsConnected

Returns a boolean value indicating if the emulation is on process.

Visual Basic Syntax

[Boolean] = TNBXmlClient.IsConnected

Delphi Syntax

[Boolean] := TNBXmlClient.IsConnected;

See also Connect and Disconnect methods.

4.2.4.4.3.6 FindEditField

Gets the specified unprotected field, searching it by its name.

Visual Basic Syntax

[TTnbField] = TNBXmlClient.FindEditField (caption As String)

Delphi Syntax

[TTnbField := TNBXmlClient.FindEditField (caption:String);

Remarks
Caption parameter specifies the name of the label associated with the field.

### 4.2.4.4.3.7 LoadFromXMLFile

Imports an XML file and then sets the `XML` property with it.

**Visual Basic Syntax**

```vbnet
TNBXmlClient.LoadFromXMLFile (XMLFile As String)
```

**Delphi Syntax**

```delphi
TNBXmlClient.LoadFromXMLFile (XMLFile: String);
```

### 4.2.4.4.3.8 SaveToXMLFile

Exports the XML code corresponding to the `TNBXmlClient` object with all the unprotected fields and the screen configuration data.

**Visual Basic Syntax**

```vbnet
TNBXmlClient.SaveToXMLFile (XMLFile As String)
```

**Delphi Syntax**

```delphi
TNBXmlClient.SaveToXMLFile (XMLFile: String);
```

### 4.2.4.4.3.9 SendAid

Sets an Attention Identifier Key to the `AidString` property and fires the `OnSendAid` event.

**Visual Basic Syntax**

```vbnet
TNBXmlClient.SendAid (AidKey As string)
```

**Delphi Syntax**

```delphi
TNBXmlClient.SendAid (AidKey: string);
```
This method takes a string with the name of the AID key as a parameter. You can specify only one key for each call to the method.

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>PF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>PF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>PF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>PF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>PF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>PA1</td>
<td>PA1 key</td>
</tr>
</tbody>
</table>

4.2.4.4.3.10 SendKeys

SendKeys can be used to send key sequences of characters (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host...
functions.

**Visual Basic Syntax**

`TNBXmlClient.SendKeys (Keys As String)`

**Delphi Syntax**

`TNBXmlClient.SendKeys (Keys:string);`

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the `Keys` parameter:

"logon applid(cics)@E"
Remarks

The execution of the SendKeys is stopped when an AID Key is sent and when this happen, the AidKey property is set with this AidKey value.

4.2.4.4.3.11 SetConnected

Sets the connection state according to the parameter received.

Visual Basic Syntax

TNBXmlClient.SetConnected [= Boolean]

Delphi Syntax

TNBXmlClient.SetConnected [:= Boolean];

4.2.4.4.4 Events

4.2.4.4.4.1 OnConnect

Occurs when the program called the SetConnected method with true, indicating that the connection has been successfully established.

See also Connect and SetConnected methods, OnConnectRemote, OnConnectionFail, OnDisconnect and OnDisconnectRemote events.

4.2.4.4.4.2 OnConnectionFail

Occurs when the program called the SetConnected method with false, indicating that the connection couldn't be established.

See also Connect method, OnConnect and OnDisconnect events.
4.2.4.4.4.3 OnConnectRemote

Occurs as a consequence of the Connect method execution. You must set the SetConnected property to True or False, according to the result of the connection to the final resource.

See also Connect and SetConnected methods, OnConnectionFail, OnDisconnect and OnDisconnectRemote events.

4.2.4.4.4 OnDisconnect

Occurs when the SetConnected property is set to False.

See also Disconnect and SetConnected methods, OnConnect, OnConnectRemote, OnDisconnect and OnDisconnectRemote event.

4.2.4.4.5 OnDisconnectRemote

Occurs as a consequence of the Disconnect method execution. You must set the SetConnected property to True or False, according to the result of the disconnection to the final resource.

See also Disconnect and SetConnected methods, OnDisconnect, OnConnect, OnConnectRemote, OnDisconnectRemote events.

4.2.4.4.6 OnScreenChange

Occurs when an valid XML is assigned, either thru XML is property or LoadFromXmlFile method.

See also XML property, OnSystemLock and OnSystemUnlock events.

4.2.4.4.7 OnSendAid

Occurs before an AID key is about to be sent.
Remarks

This event is fired when a SendAid or SendKeys methods or the AidKey property are used. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator component.

See also SendAid, SendKeys methods and AidKey property.

4.2.4.4.4.8 OnSystemLock

Occurs when the XML stream changes to a locked state.

Remarks

During this state, sending data isn't possible until the OnSystemUnlock event is fired.

See also OnSystemUnlock event.

4.2.4.4.4.9 OnSystemUnlock

Occurs when the XML stream changes to an unlocked state.

Remarks

During this state, sending data is possible until the OnSystemLock event occurs.

See also OnSystemLock event.

4.2.4.5 TTnbXmlVirtual Component

4.2.4.5.1 TNBXmlVirtual Component

This component allows you to create a simulated host application by combining XML-screen files and code to drive the screen navigation. XML-screen files can be taken using Development Lab.

Properties

- AidKey
4.2.4.5.2 Properties

4.2.4.5.2.1 AidKey

Sets the Attention Identifier Key and executes the \texttt{SendAid} method. Modified fields can be input fields (unprotected) or, sometimes protected fields, having the property Modified set to True.

\textbf{Visual Basic Syntax}

\begin{verbatim}
TNBXmlVirtual.AIDKey \[= \texttt{TNBAid}\]
\end{verbatim}

\textbf{Delphi Syntax}

\begin{verbatim}
TNBXmlVirtual.AIDKey \[:= \texttt{TNBAid}\];
\end{verbatim}
It can take any of AidKey constant values:

<table>
<thead>
<tr>
<th>Constant Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>akNone</td>
<td>No Action</td>
</tr>
<tr>
<td>akEnter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>akClear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>akATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>akSysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>akPF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>akPF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>akPF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>akPF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>akPF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>akPF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>akPF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>akPF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>akPF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>akPF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>akPF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>akPF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>akPF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>akPF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>akPF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>akPF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>akPF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>akPF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>akPF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>akPF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>akPF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>akPF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>akPF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>akPF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>akPA1</td>
<td>PA1 key</td>
</tr>
<tr>
<td>akReset</td>
<td>Reset key</td>
</tr>
<tr>
<td>akPgUp</td>
<td>Page Up key</td>
</tr>
<tr>
<td>akPgDown</td>
<td>Page Down key</td>
</tr>
<tr>
<td>akPgLeft</td>
<td>Page Left key</td>
</tr>
<tr>
<td>akPgRight</td>
<td>Page Right key</td>
</tr>
<tr>
<td>akPrint</td>
<td>Print key</td>
</tr>
<tr>
<td>akHelp</td>
<td>Help key</td>
</tr>
</tbody>
</table>

See also Aid constants.
4.2.4.5.2.2 BaseDirectory

Gets/sets the base directory from where Screen XML files will be loaded.

**Visual Basic Syntax**

```
TNBXmlVirtual.BaseDirectory [= BaseDirectory As String]
```

**Delphi Syntax**

```
TNBXmlVirtual.BaseDirectory [= BaseDirectory:String];
```

**Remarks**

BaseDirectory is used to automatically load screen files when you press PgDown/PgUp keys.

See also [OnNewScreen](#) event and [StartFilename](#) property.

---

4.2.4.5.2.3 CurrentScreen

Gets the name of current XML filename used to render the current screen.

**Visual Basic Syntax**

```
TNBXmlVirtual.CurrentScreen [= CurrentScreen As String]
```

**Delphi Syntax**

```
TNBXmlVirtual.CurrentScreen [= CurrentScreen:String];
```

---

4.2.4.5.2.4 ExcludeEmptyFields

Sets/gets the possibility of excluding all empty fields.

**Visual Basic Syntax**

```
TNBXmlVirtual.ExcludeEmptyFields [= ExcludeEmptyFields As Boolean]
```

**Delphi Syntax**
4.2.4.5.2.5 FieldSplitting

Determines the format of Fields in HostFields and EditFields collections.

**Visual Basic Syntax**

```vbnet
TNBXmlVirtual.FieldSplitting [= FieldSplitting As Boolean]
```

**Delphi Syntax**

```delphi
TNBXmlVirtual.FieldSplitting [= FieldSplitting:Boolean];
```

**Remarks**

FieldSplitting property controls how fields in HostFields and EditFields collections are constructed. When active, fields that go beyond the maximun of display column are splitted in two or more fields.

Default is `true`.

4.2.4.5.2.6 StartFilename

Gets/Sets the start XML File.

**Visual Basic Syntax**

```vbnet
TNBXmlVirtual.StartFileName [= StartFileName As String]
```

**Delphi Syntax**

```delphi
TNBXmlVirtual.StartFileName [= StartFileName:String];
```

**Remarks**

When you call the connect method the StartFilename will be the one that will be used to provide the content for the first screen. BaseDirectory will be set to the directory where StartFilename belongs.
See also OnNewScreen event and BaseDirectory property.

4.2.4.5.2.7 Text

Returns the entire screen.

**VB Syntax**

```
[String =] TNBXmIVirtual.Text
```

**Delphi Syntax**

```
[String =] TNBXmIVirtual.Text;
```

4.2.4.5.2.8 XML

When assigning an XML code, it generates a TnbFields collection. Reading this property, generates and returns an XML code from unprotected TnbFields, cursor location and AID Key.

**Visual Basic Syntax**

```
TNBXmIVirtual.XML [= String]
```

**Delphi Syntax**

```
TNBXmIVirtual.XML [:= String];
```

4.2.4.5.3 Methods

4.2.4.5.3.1 Connect

The Connect method initializes the connection sequence process. It loads the StartFilename XML File.

**Visual Basic Syntax**

```
TNBXmIVirtual.Connect
```
4.2.4.5.3.2 Disconnect

The Disconnect method ends the connection sequence process.

**Visual Basic Syntax**

```vbnet
TNBXmlVirtual.Disconnect
```

**Delphi Syntax**

```delphi
TNBXmlVirtual.Disconnect;
```

See also Connect method and OnDisconnect event.

4.2.4.5.3.3 IsConnected

Returns a boolean value indicating if the emulation is on process.

**VB Syntax**

```vbnet
[Boolean =] TNBXmlVirtual.IsConnected
```

**Delphi Syntax**

```delphi
[boolean :=] TNBXmlVirtual.IsConnected;
```

See also Connect and Disconnect methods.
4.2.4.5.3.4 IsLocked

Returns a boolean value indicating if the Host is in a locked state.

**VB Syntax**

\[
[\text{Boolean} =] \ TNB XmIVirtual.\text{IsLocked}
\]

**Delphi Syntax**

\[
[\text{boolean :} =] \ TNB XmIVirtual.\text{IsLocked};
\]

4.2.4.5.3.5 IsScreenEmpty

Returns a boolean value indicating in the screen has all the fields with no content or in filled with space characters.

**VB Syntax**

\[
[\text{Boolean} =] \ TNB XmIVirtual.\text{IsScreenEmpty}
\]

**Delphi Syntax**

\[
[\text{boolean :} =] \ TNB XmIVirtual.\text{IsScreenEmpty};
\]

4.2.4.5.3.6 GetText

GetText returns the text buffer of a portion of the current screen. You can receive the attribute or extended attribute of each field together to the character data.

**VB Syntax**

\[
[\text{String} =] \ TNB XmIVirtual.\text{GetText} (\text{StartPos As Integer, EndPos As Integer}, [\text{Attr As Boolean} = \text{False}], [\text{Eab As Boolean} = \text{False}])
\]

\[
[\text{String} =] \ TNB XmIVirtual.\text{GetText} (\text{Row As Integer, Col As Integer, Len as Integer}, [\text{Attr As Boolean} = \text{False}], [\text{Eab As Boolean} = \text{False}])
\]

**Delphi Syntax**

\[
[\text{string :} =] \ TNB XmIVirtual.\text{GetText} (\text{int StartPos, int StartPos}, [\text{bool Attr = False}], [\text{bool Eab = False}]);
\]

\[
[\text{string :} =] \ TNB XmIVirtual.\text{GetText} (\text{int Row, in Col, int Len}, [\text{bool Attr = False}], [\text{bool}]
\]
Eab = False));

Remarks

Start position and the end position of the text buffer to retrieve can be set using StartPos and EndPos parameters or Row,Col and Len parameters.

The Attr parameter will indicate to GetText that includes the field attribute information. The Attr describes the field properties and occupies the first character position of each field in the character buffer and on the screen. This character is in HLLAPI-Compliance format.

The Eab parameter will indicate to GetText that includes the extended attribute information. The Eab is associated with individual characters. Each character in the buffer (except for fields attributes) has a character attribute. In the returned String, each character will be preceded by his Eab value. This character is in HLLAPI-Compliance format.

4.2.4.5.3.7 LoadScreen

Loads an XML file to provide with new screen content.

VB Syntax

[Boolean] = TNBXmIVirtual.LoadScreen (xmlFileName As string)

Delphi Syntax

[boolean :=] TNBXmIVirtual.LoadScreen (xmlFileName:string);

4.2.4.5.3.8 Press

Sets an Attention Identifier Key and sends the modified fields to the Mainframe. Modified fields can be input fields (unprotected) or sometimes protected fields, having the property Modified set to True.

VB Syntax

[Boolean] = TNBXmIVirtual.Press (Value As string)

Delphi Syntax

[boolean :=] TNBXmIVirtual.Press (Value:string);
This method works like the AidKey property, but it takes a string with the name of the AID key as a parameter. You can specify only one key for each call to the method.

The following table lists the possible string values:

<table>
<thead>
<tr>
<th>String Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter</td>
<td>ENTER key</td>
</tr>
<tr>
<td>Clear</td>
<td>CLEAR key</td>
</tr>
<tr>
<td>ATTN</td>
<td>Attention key</td>
</tr>
<tr>
<td>SysReq</td>
<td>System Request key</td>
</tr>
<tr>
<td>PF1</td>
<td>PF01 key</td>
</tr>
<tr>
<td>PF2</td>
<td>PF02 key</td>
</tr>
<tr>
<td>PF3</td>
<td>PF03 key</td>
</tr>
<tr>
<td>PF4</td>
<td>PF04 key</td>
</tr>
<tr>
<td>PF5</td>
<td>PF05 key</td>
</tr>
<tr>
<td>PF6</td>
<td>PF06 key</td>
</tr>
<tr>
<td>PF7</td>
<td>PF07 key</td>
</tr>
<tr>
<td>PF8</td>
<td>PF08 key</td>
</tr>
<tr>
<td>PF9</td>
<td>PF09 key</td>
</tr>
<tr>
<td>PF10</td>
<td>PF10 key</td>
</tr>
<tr>
<td>PF11</td>
<td>PF11 key</td>
</tr>
<tr>
<td>PF12</td>
<td>PF12 key</td>
</tr>
<tr>
<td>PF13</td>
<td>PF13 key</td>
</tr>
<tr>
<td>PF14</td>
<td>PF14 key</td>
</tr>
<tr>
<td>PF15</td>
<td>PF15 key</td>
</tr>
<tr>
<td>PF16</td>
<td>PF16 key</td>
</tr>
<tr>
<td>PF17</td>
<td>PF17 key</td>
</tr>
<tr>
<td>PF18</td>
<td>PF18 key</td>
</tr>
<tr>
<td>PF19</td>
<td>PF19 key</td>
</tr>
<tr>
<td>PF20</td>
<td>PF20 key</td>
</tr>
<tr>
<td>PF21</td>
<td>PF21 key</td>
</tr>
<tr>
<td>PF22</td>
<td>PF22 key</td>
</tr>
<tr>
<td>PF23</td>
<td>PF23 key</td>
</tr>
<tr>
<td>PF24</td>
<td>PF24 key</td>
</tr>
<tr>
<td>PA1</td>
<td>PA1 key</td>
</tr>
</tbody>
</table>

See also AidKey property.
4.2.4.5.3.9 PressAndWait

(add to list)

4.2.4.5.3.10 SetConnected

Sets the connection state according to the parameter received.

**VB Syntax**

```
TNBXmlVirtual.SetConnected (Value As Boolean)
```

**Delphi Syntax**

```
TNBXmlVirtual.SetConnected (value:boolean);
```

4.2.4.5.3.11 SetText

SetText allows to put text information into edit buffer.

**VB Syntax**

```
TNBXmlVirtual.SetText (StartPos As Integer, Text As String)
```

```
TNBXmlVirtual.SetText (Row As Integer; Col As Integer; Text As String)
```

**Delphi Syntax**

```
TNBXmlVirtual.SetText (StartPos:integer; Text:string)
```

```
TNBXmlVirtual.SetText (Row:integer; Col:integer; Text:string)
```

**Remarks**

SetText allows to put text information into edit buffer, starting according to position StartPos or Row and Col parameters.

4.2.4.5.3.12 Type

Type can be used to send key sequences to the mainframe (including AID keys). By using special codes you can send several kinds of keys. These codes consist of an Escape character (default is "@") and a mnemonic code that corresponds to the supported host functions.
VB Syntax

TNBXmlVirtual.Type (DataString As String)

Delphi Syntax

TNBXmlVirtual.Type (DataString:string);

The following table lists the functions keys and its corresponding codes.

<table>
<thead>
<tr>
<th>Code Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>@A@Q</td>
<td>Attention</td>
</tr>
<tr>
<td>@&lt;</td>
<td>Backspace</td>
</tr>
<tr>
<td>@B</td>
<td>BackTab (Left Tab)</td>
</tr>
<tr>
<td>@C</td>
<td>Clear</td>
</tr>
<tr>
<td>@E</td>
<td>Enter</td>
</tr>
<tr>
<td>@F</td>
<td>Erase Field</td>
</tr>
<tr>
<td>@A@Q</td>
<td>Sys Request</td>
</tr>
<tr>
<td>@T</td>
<td>Tab (Right Tab)</td>
</tr>
<tr>
<td>@x</td>
<td>PA1</td>
</tr>
<tr>
<td>@y</td>
<td>PA2</td>
</tr>
<tr>
<td>@z</td>
<td>PA3</td>
</tr>
<tr>
<td>@1</td>
<td>PF1</td>
</tr>
<tr>
<td>@2</td>
<td>PF2</td>
</tr>
<tr>
<td>@3</td>
<td>PF3</td>
</tr>
<tr>
<td>@4</td>
<td>PF4</td>
</tr>
<tr>
<td>@5</td>
<td>PF5</td>
</tr>
<tr>
<td>@6</td>
<td>PF6</td>
</tr>
<tr>
<td>@7</td>
<td>PF7</td>
</tr>
<tr>
<td>@8</td>
<td>PF8</td>
</tr>
<tr>
<td>@9</td>
<td>PF9</td>
</tr>
<tr>
<td>@a</td>
<td>PF10</td>
</tr>
<tr>
<td>@b</td>
<td>PF11</td>
</tr>
<tr>
<td>@c</td>
<td>PF12</td>
</tr>
<tr>
<td>@d</td>
<td>PF13</td>
</tr>
<tr>
<td>@e</td>
<td>PF14</td>
</tr>
<tr>
<td>@f</td>
<td>PF15</td>
</tr>
<tr>
<td>@g</td>
<td>PF16</td>
</tr>
<tr>
<td>@h</td>
<td>PF17</td>
</tr>
<tr>
<td>@i</td>
<td>PF18</td>
</tr>
</tbody>
</table>

Here is an example of the Keys parameter:

"logon applid(cics)@E"

Remarks
The execution of the SendKeys is stopped when an AID Key is sent to the Host.

See also **AIDKey** property.

### 4.2.4.5.4 Events

#### 4.2.4.5.4.1 OnConnect

Occurs when the **Connect** method is called and the **StartFilename** is loaded.

See also **Connect** method, **OnConnectionFail** and **OnDisconnect** events.

#### 4.2.4.5.4.2 OnConnectionFail

Occurs when the program fails to load the **StartFilename** XML File.

See also **Connect** method, **OnConnect** and **OnDisconnect** events.

#### 4.2.4.5.4.3 OnDisconnect

Occurs when the **Disconnect** method is called.

See also **Disconnect** method, **OnConnect** and **OnDisconnect** event.

#### 4.2.4.5.4.4 OnNewScreen

Occurs when a new XML file should be loaded.

**Remarks**

When this event is fired, you can analyze **AidKey** to determine the next screen you should load using **LoadScreen** method.

See also **LoadScreen** method.
4.2.4.5.4.5 OnScreenChange

Occurs when a valid XML is assigned, either thru XML is property or LoadFromXmlFile method.

See also XML property, OnSystemLock and OnSystemUnlock events.

4.2.4.5.4.6 OnSendAid

Occurs before an AID key is about to be sent.

Remarks

This event is fired when a SendAid or SendKeys methods or the AidKey property are used. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator component.

See also SendAid, SendKeys methods and AidKey property.

4.2.4.5.4.7 OnSystemLock

Occurs when the XML stream changes to a locked state.

Remarks

During this state, sending data isn't possible until the OnSystemUnlock event is fired.

See also OnSystemUnlock event.

4.2.4.5.4.8 OnSystemUnlock

Occurs when the XML stream changes to an unlocked state.

Remarks
During this state, sending data is possible until the **OnSystemLock** event occurs.

*See also* **OnSystemLock** event.

### 4.2.5 Trace Services

Find in the topics within this section a complete reference for the TNBridge Host Integration Pack Delphi Trace Services.

#### 4.2.5.1 Setting up Trace Services

To trace a TN Bridge application you need to setup the Trace Agent built into the **TNBTrace** component.

The following steps shows you how to setup the Trace Services to work with your application:

- Drop a **TNBTrace** component into a form.
- Set the **TnbCom** property of the trace component to the Telnet component:

  ```
  TNBTrace1.TnbCom := Telnet
  ```

- Set the TCP/IP listening port to the **Port** property (at design time or at run time):

  ```
  TNBTrace1.Port := 1024
  ```

- Set the **Active** property to true. When doing this at design-time, the activation takes place at run-time:

  ```
  TNBTrace1.Active := True
  ```

- Run your application.

- Each time the telnet component connects to a mainframe, a trace file is generated with a connection identifier as its name and extension the ".hst". This file will be stored into the folder specified in TNBTrace Directory property or in windows temporary files folder. By default, this files will be removed when the trace becomes inactive, but you can change this behavior setting the **PurgeTraceFiles** property to **False**.
For on-line monitoring:

- Open the TN Bridge Trace Viewer and setup a connection pointing to the machine IP address where your TN Bridge application is running. Also set the TCP Port to the listening port specified in your TNBTrace component. This is the Connections Settings dialog box where you can complete the information:

```
Connections Settings
Name: Mi queido HCT
Host: 127.0.0.1
Port: 1024
```

- Click the connect button. A dialog box with the available connections will be shown. Choose one from the list and click the Connect button.

For off-line monitoring:

- Open the TN Bridge Trace Viewer and choose the file menu and open menu item.
Select the trace file previously generated by your TN Bridge application. (Remember that this file will be available only if you set the TNBTrace’s PurgeTraceFiles property to False. Also, it’s strongly recommended to set the Directory property to an existing a known folder).

4.2.5.2 TN Bridge Trace Server

4.2.5.2.1 TNBTrace Component

This component implements a Trace Agent. It allows real-time monitoring of Telnet events, analyses mainframe’s screen information and TNBSync methods calls.

Properties

- Active
- Directory
- HidePasswordFields
- Port
- PurgeTraceFiles
- TnbCom
- Visible

Methods

- HideTrace
- ShowTrace
- Trace

4.2.5.2.2 Properties

4.2.5.2.2.1 Active

Activates or deactivates Trace Agent functionality.

Visual Basic Syntax

TNBTrace.Active [= Boolean]

Delphi Syntax

TNBTrace.Active; [= Boolean]

Remarks
If you assign the value of this property at design time, it acts only as the initial value for run-time.

Default value is **False**.

### 4.2.5.2.2.2 Directory

Sets/gets the folder directory name where the trace files will be saved.

**Visual Basic Syntax**

```vbnet
TNBTrace.Directory [= String]
```

**Delphi Syntax**

```delphi
TNBTrace.Directory; [= String]
```

**Remarks**

If you don't specify a directory name, the trace files will be saved into the windows temporary file directory.

### 4.2.5.2.2.3 HidePasswordFields

Enables or disables the password masking mode for hiding data of password fields.

**Visual Basic Syntax**

```vbnet
TNBTrace.HidePasswordFields [= Boolean]
```

**Delphi Syntax**

```delphi
TNBTrace.HidePasswordFields; [= Boolean]
```

**Remarks**

When this property is set to **True**, password fields are masked with asterisks.

Default value is **True**.
4.2.5.2.2.4 Port

Sets/gets the listening TCP port number for the current application process. The TNBTrace component works at process level, generating independent traces on a connection basis.

**Visual Basic Syntax**

TNBTrace.Port [= Integer]

**Delphi Syntax**

TNBTrace.Port; [= Integer]

**Remarks**

The default port value is 1024.

4.2.5.2.2.5 PurgeTraceFiles

Allows to purge trace files when the server becomes inactive.

**Visual Basic Syntax**

TNBTrace.PurgeTraceFiles [= Boolean]

**Delphi Syntax**

TNBTrace.PurgeTraceFiles; [= Boolean]

**Remarks**

By default, trace information is kept into temporary files. When the Active property is set to False or the process is finished, these files are deleted.

Default value is True.
4.2.5.2.2.6 TnbCom

Sets the TNB3270 or TNB5250 component as the telnet client control.

**Visual Basic Syntax**

```vbnet
TNBTrace.TnbCom [= TNBnnnn]
```

**Delphi Syntax**

```delphi
TNBTrace.TnbCom; [= TNBnnnn]
```

**Remarks**

You must set this property to allow TNBTrace component to work properly.

4.2.5.2.2.7 Visible

Activates or deactivates Trace Viewer window. It can be used at the same time you are developing an Integration Pack application, helping about host events that happen when application code is running. Trace Viewer window helps to follow those host events as you modify application's code.

**Visual Basic Syntax**

```vbnet
TNBTrace.Visible [= Boolean]
```

**Delphi Syntax**

```delphi
TNBTrace.Visible; [= Boolean]
```

**Remarks**
If you set the value of this property at design time to True, Trace Viewer window is shown before you run Integration Pack application, and will be prepared to show hosts events at application run-time. Default value is **False**.

See also **ShowTrace** and **HideTrace** methods.

### 4.2.5.2.3 Methods

#### 4.2.5.2.3.1 HideTrace

Hides Trace Viewer window. It can be used at the same time you are developing an Integration Pack application, helping you with host events that happen when application code is running.

**Visual Basic Syntax**

TNBTrace.HideTrace

**Delphi Syntax**

TNBTrace.HideTrace;

**Remarks**

This method is used in developing Integration Pack's application process.

See also **ShowTrace** method and **Visible** property.

#### 4.2.5.2.3.2 ShowTrace

Shows Trace Viewer window. It can be used at the same time you are developing an Integration Pack application, helping about host events that happen when application code is running.

**Visual Basic Syntax**

TNBTrace.ShowTrace

**Delphi Syntax**

TNBTrace.ShowTrace;
Remarks

This method is used in developing Integration Pack's application process.

See also HideTrace method and Visible property.

4.2.5.2.3.3 Trace

Allows to insert custom string data into the trace file.

Visual Basic Syntax

`TNBTrace.Trace (Message As String)`

Delphi Syntax

`TNBTrace.Trace (Message: string);`

Remarks

This method can be used for debugging purposes by making an entry with String content in the trace file.

5 OHIO Reference

- Introduction to OHIO
- Programming Reference

5.1 Introduction

Open Host Interface Objects (OHIO) is an IBM and Attachmate inspired object-oriented host access API for software implementing tn3270 and tn5250 protocols. Vendor neutral and submitted to the IETF in 1998 as an Internet standard.

OHIO address the need for a standardized advanced programming interface to the host data. OHIO does not modify the TN3270/TN5250 protocol or datastream but instead
provides a common access method to that data once it arrives at the client. It uses an Object Oriented approach to divide the data into logical objects, and provides methods on those objects to allow standard access to the data.

Some OHIO's advantages are:

- You can run an OHIO application without having an emulation session running.
- OHIO has all the benefits of the object oriented programming paradigm because it was designed as an object oriented API.
- You can concentrate on the application functions, without worrying about structure packing details or parameter command codes.

In this chapter we detail:

- Ohio containment hierarchy
- Ohio inheritance hierarchy

5.1.1 Ohio containment hierarchy

The following is the Ohio containment hierarchy:
Additional utility classes:
  - OhioPosition

Note: "1" based counting is used throughout this document for both positions (the first position on the screen is position 1, not position 0) and sizes (the first item in a collection is item 1, not item 0).

5.1.2 Ohio inheritance hierarchy

The Ohio inheritance hierarchy is:

```
OHIO: Base Class
     ▼
    /   \
Manager  Sessions  Session  OIA  Fields  Field  Position
```

5.2 Programming Reference

In this chapter we describe the following topics:

- Ohio Base Class
- Class Definition
- Constants
5.2.1 Ohio Base Class

This is the base class for all Ohio classes. Contains all common Ohio methods and properties.

See also Ohio Base Class Properties and Methods.

5.2.1.1 Syntax Conventions

The following text formats are used throughout the Components Reference:

For each property and method:

*Italic text* Denotes an object, in this case a TNBridge control.

*Bold text* Denotes a property or a method names.

 [= value] or [:= value] Denotes values to be assigned.

[ value =] or [value :=] Denotes return values.

(TimeOut) Denotes a parameter to be passed to a method.

([TimeOut]) Denotes an optional parameter to be passed to a method.

Note: When we refer to OhioObject we are indicating that this object can be any of the objects that belong to the Ohio Base Class.

5.2.1.2 Methods

- CreateOhioPosition

5.2.1.2.1 CreateOhioPosition

Creates an OhioPosition object.

Visual Basic Syntax
Delphi Syntax

\[\text{OhioObject.CreateOhioPosition(row As Integer, col As Integer)};\]

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>The row coordinate</td>
</tr>
<tr>
<td>col</td>
<td>The column coordinate</td>
</tr>
</tbody>
</table>

#### 5.2.1.3 Properties

- **OhioVersion**
- **VendorName**
- **VendorObject**
- **VendorProductVersion**

##### 5.2.1.3.1 OhioVersion

Returns the Ohio Version level for this implementation (OHIO 1.00).

**Visual Basic Syntax**

\[\text{OhioObject.OhioVersion}\]

**Delphi Syntax**

\[\text{OhioObject.OhioVersion};\]

##### 5.2.1.3.2 VendorName

Returns the name of the vendor providing this OHIO implementation. Format is vendor defined.

**Visual Basic Syntax**

\[\text{OhioObject.VendorName}\]
[String =] OhioObject.**VendorName**

**Delphi Syntax**

[String :=] OhioObject.**VendorName**;

### 5.2.1.3.3 VendorObject

Returns the VendorObject for the current session. This property returns different objects according to the object who calls it.

**Visual Basic Syntax**

[T Tennet =>] Screen.**VendorObject**

[T Tennet =>] Session.**VendorObject**

[T OIA =>] OIA.**VendorObject**

[T Sessions =>] Sessions.**VendorObject**

[T Field =>] Field.**VendorObject**

**Delphi Syntax**

[T Tennet :=] Screen.**VendorObject**;

[T Tennet :=] Session.**VendorObject**;

[T OIA :=] OIA.**VendorObject**;

[T Sessions :=] Sessions.**VendorObject**;

[T Field :=] Field.**VendorObject**;

### 5.2.1.3.4 VendorProductVersion

Indicates the vendor product version that is providing the OHIO implementation. Format is vendor specific.

**Visual Basic Syntax**

[String =] OhioObject.**VendorProductVersion**
Delphi Syntax


5.2.2 Class Definition

- OhioManager
- OhioSessions
- OhioSession
- OhioScreen
- OhioFields
- OhioField
- OhioOIA
- OhioPosition

5.2.2.1 Syntax Conventions

The following text formats are used throughout the Components Reference:

For each property and method:

*Italic text* Denotes an object, in this case a TNBridge control.

*Bold text* Denotes a property or a method names.

[= value] or [:= value] Denotes values to be assigned.

[value =] or [value :=] Denotes return values.

(TimeOut) Denotes a parameter to be passed to a method.

([TimeOut]) Denotes an optional parameter to be passed to a method.

5.2.2.2 OhioManager

The OhioManager is the central repository for access to all OHIO sessions. It contains a list of all OhioSession objects available on this system.

See also OhioManager Properties and Methods.
5.2.2.2.1 Methods

- **OpenSession**
- **CloseSession**

5.2.2.2.1.1 OpenSession

Returns an OhioSession object based on the search parameters provided.

**Visual Basic Syntax**

\[
\text{TtnbOhioSession} = \text{Manager.OpenSession}(
\text{ConfigurationResource} \text{ As Integer, SessionName As String})
\]

\[
\text{TtnbOhioSession} = \text{Manager.OpenSession}(
\text{ConfigurationResource} \text{ As TTnbComApi, SessionName As String})
\]

\[
\text{TtnbOhioSession} = \text{Manager.OpenSession}(
\text{ConfigurationResource} \text{ As String, SessionName As String})
\]

**Delphi Syntax**

\[
\text{TtnbOhioSession} := \text{Manager.OpenSession}(
\text{ConfigurationResource:integer; SessionName:string})
\]

\[
\text{TtnbOhioSession} := \text{Manager.OpenSession}(
\text{ConfigurationResource:TTnbComApi; SessionName:string})
\]

\[
\text{TtnbOhioSession} := \text{Manager.OpenSession}(
\text{ConfigurationResource, SessionName:string})
\]

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationResource</td>
<td>A vendor specific string used to provide configuration information</td>
</tr>
<tr>
<td>SessionName</td>
<td>The unique name associated with an OhioSession</td>
</tr>
</tbody>
</table>

The parameters are used as follows:

Is ConfigurationResource provided?
Yes - Is SessionName provided?
  Yes - Is OhioSession object with matching SessionName available on the system?
  Yes - Error, attempting to create an
OhioSession object with a non-unique SessionName.
No - Create an OhioSession object using SessionName and ConfigurationResource.
No - Start a new OhioSession using ConfigurationResource and generating a new SessionName.
No - Is SessionName provided?
   Yes - Is OhioSession object with matching SessionName available on the system?
   Yes - Return identified OhioSession object.
   No - Return null.
No - Return null.

5.2.2.2.1.2 CloseSession

Closes an OhioSession object. The OhioSession is considered invalid and is removed from the list of OhioSession objects. You can use this method with the following syntaxes:

**Visual Basic Syntax**

Manager.CloseSession(SessionObject As TTnbOhioSession)
Manager.CloseSession(SessionName As String)

**Delphi Syntax**

Manager.CloseSession(SessionObject:TTnbOhioSession);
Manager.CloseSession(SessionName:string);

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionObject</td>
<td>The OhioSession to close.</td>
</tr>
<tr>
<td>SessionName</td>
<td>The SessionName of the OhioSession to close.</td>
</tr>
</tbody>
</table>

5.2.2.2.2 Properties

- **Sessions**
5.2.2.2.2.1 Sessions

Returns an OhioSessions object containing the OhioSession objects available on this system. This list of objects is a static snapshot at the time the OhioSessions object is created.

**Visual Basic Syntax**

\[\text{OhioSessions} = \] Manager.\text{Sessions} \\

**Delphi Syntax**

\[\text{OhioSessions} := \] Manager.\text{Sessions;} \\

5.2.2.3 OhioSessions

Contains a collection of OhioSession objects. This list is a static snapshot of the list of OhioSession objects available at the time of the snapshot.

**See also** OhioSessions Properties and Methods.

5.2.2.3.1 Methods

- **AddSession**
- **CloseSession**

5.2.2.3.1.1 AddSession

Adds a session to the Sessions collection based on the specified configuration file.

**Visual Basic Syntax**

\[\text{TTnbOhioSession} = \] Sessions.\text{AddSession}(\text{ConfigurationResource As String, SessionName As String}) \]
[TTnbOhioSession :=] Sessions.AddSession(ConfigurationResource As Integer, SessionName As String)

[TTnbOhioSession :=] Sessions.AddSession(ConfigurationResource As TTnbComApi, SessionName As String)

**Delphi Syntax**

[TTnbOhioSession :=] Sessions.AddSession(ConfigurationResource As TTnbComApi, SessionName := string);

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigurationResource</td>
<td>Full path of the Configuration File (.XML) to use for host connection information. This parameter can be a string, an integer or a TTnbComApi type.</td>
</tr>
<tr>
<td>SessionName</td>
<td>Session's name.</td>
</tr>
</tbody>
</table>

5.2.2.3.1.2 CloseSession

Closes an OhioSession object. The OhioSession is considered invalid and is removed from the list of OhioSession objects. You can use this methods with the following syntaxes:

**Visual Basic Syntax**

Session.CloseSession(SessionObject As TTnbOhioSession)

Session.CloseSession(SessionName As String)

**Delphi Syntax**

Sessions.CloseSession(SessionObject:TTnbOhioSession);

Sessions.CloseSession(SessionName:string);

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionObject</td>
<td>The OhioSession to close.</td>
</tr>
</tbody>
</table>
5.2.2.3.2 Properties

- **Count**
- **Item**
- **Manager**

5.2.2.3.2.1 Count

Returns the number of OhioSessions objects contained in this collection (the number of Sessions currently opened).

**Visual Basic Syntax**

```vbnet
[Integer =] Sessions.Count
```

**Delphi Syntax**

```delphi
[Integer :=] Sessions.Count;
```

5.2.2.3.2.2 Item

Returns the OhioSessions object at the given index. "One based" indexing is used in all Ohio collections. For example, the first OhioSession in this collection is at index 1.

**Visual Basic Syntax**

```vbnet
[TTnbOhioSession] = Sessions.Item(index As Variant)
```

**Delphi Syntax**

```delphi
[TTnbOhioSession] := Sessions.Item(index:Variant);
```

**Parameters**

| Parameters | Description |
5.2.2.3.2.3 Manager

Returns the OhioManager object for the current session.

**Visual Basic Syntax**

Sessions.Manager [= OhioManager]

**Delphi Syntax**

Sessions.Manager; [= OhioManager]

5.2.2.4 OhioSession

The OhioSession represents a host session.

*See also* OhioSession *Properties, Methods* and *Events*.

5.2.2.4.1 Methods

- **Connect**
- **Disconnect**
- **WaitForConnect**
- **WaitForDisconnect**

5.2.2.4.1.1 Connect

Starts the communications link to the host.

**Visual Basic Syntax**

Session.Connect
5.2.2.4.1.2 Disconnect

Stops the communications link to the host.

**Visual Basic Syntax**

```vbnet
Session.Disconnect
```

**Delphi Syntax**

```delphi
Session.Disconnect;
```

**Parameters**

None.

5.2.2.4.1.3 WaitForConnect

This method waits until the telnet connection is successfully opened, or the timeout period expires.

**Visual Basic Syntax**

```vbnet
[Boolean =] Session.WaitForConnect([TimeOut] As Integer)
```

**Delphi Syntax**

```delphi
[WordBool :=] Session.WaitForConnect([TimeOut]: Integer);
```

**Remarks**
The Timeout parameter is optional. This method returns True if the wait was successful and False if the Timeout period has expired. The TimeOut parameter is measured in milliseconds.
If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

5.2.2.4.1.4 WaitForDisconnect

This method waits until the telnet connection is successfully closed, or the timeout period expires.

Visual Basic Syntax

[Boolean =] Session.WaitForDisconnect([TimeOut] As Integer)

Delphi Syntax

[WordBool :=] Session.WaitForDisconnect([TimeOut]: Integer);

Remarks

The Timeout parameter is optional. This method returns True if the wait was successful and False if the Timeout period has expired. The TimeOut parameter is measured in milliseconds.
If you pass a timeout value in the Timeout parameter of this method, the value of the property WaitTimeout will not have any effect.

5.2.2.4.2 Properties

- ConfigurationResource
- Connected
- Manager
- Screen
- SessionName
- SessionState
- SessionType
- Trace
5.2.2.4.2.1 ConfigurationResource

Indicates the ConfigurationResource for this OhioSession object.

**Visual Basic Syntax**

[String =] Session.ConfigurationResource

**Delphi Syntax**


**Remarks**

This property returns the full path of the configuration resource file for the current session.

5.2.2.4.2.2 Connected

Indicates whether this OhioSession object is connected to a host.

**Visual Basic Syntax**

[Boolean =] Session.Connected

**Delphi Syntax**

[Boolean :=] Session.Connected;

**Remarks**

True means connected, False means not connected.

5.2.2.4.2.3 Manager

Returns the OhioManager object for the current session.

**Visual Basic Syntax**

Session.Manager [= OhioManager]

**Delphi Syntax**
5.2.4.2.4 Screen

Returns the OhioScreen object for the current session.

**Visual Basic Syntax**

`[OhioScreen =] Session.Screen`

**Delphi Syntax**

`[OhioScreen :=] Session.Screen;`

5.2.4.2.5 SessionName

Returns the SessionName for this OhioSession object.

**Visual Basic Syntax**

`[String =] Session.SessionName`

**Delphi Syntax**

`[String :=] Session.SessionName;`

**Remarks**

The SessionName is unique among all instances of OhioSession.

5.2.4.2.6 SessionState

Returns the SessionState for the Session object.

**Visual Basic Syntax**

`[String =] Sessions.SessionState`
Delphi Syntax

[String :=] Session.SessionState;

5.2.2.4.2.7 SessionType

Returns the SessionType for this OhioSession object.

Visual Basic Syntax

[Integer =>] Session.SessionType

Delphi Syntax

[Integer :=] Session.SessionType;

5.2.2.4.2.8 Trace

Sets the TNBXTrace Control as its trace agent.

Visual Basic Syntax

Session.Trace [= TTnbTrace]

Delphi Syntax

Session.Trace; [= TTnbTrace]

Remarks

You must set this property to get trace information from TNBXSycn control.

5.2.2.4.3 Events

- OnConnect
- OnConnectionFail
- OnDisconnect
• **OnScreenChange**
  
  • **OnSendAid**
  
  • **OnSessionState**

5.2.2.4.3.1 **OnConnect**

  Occurs after a successfully connection to the server is established.

5.2.2.4.3.2 **OnConnectionFail**

  Occurs after the connection to the server fails.

5.2.2.4.3.3 **OnDisconnect**

  Occurs after a disconnection of the server.

5.2.2.4.3.4 **OnScreenChange**

  This event is generated whenever the virtual screen is modified.

5.2.2.4.3.5 **OnSendAid**

  Occurs before an Aid key is to be sent.

  **Remarks**

  This event is fired when a **SendAid** or **SendKeys** methods are used to send an Aid Key and data to the mainframe. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator control.
5.2.2.4.3.6 OnSessionState

Occurs when a session-state change takes place.

Remarks

A session state can be ssNoSession, ssSSCPLU and ssLULU. Any transition between the states fires this event.

5.2.2.5 OhioScreen

OhioScreen encapsulates the host presentation space. The presentation space is a virtual screen which contains all the characters and attributes that would be seen on a traditional emulator screen. This virtual screen is the primary object for text-based interactions with the host. The OhioScreen provides methods that manipulate text, search the screen, send keystrokes to the host, and work with the cursor.

An OhioScreen object can be obtained from the Screen property of an instance of OhioSession.

The raw presentation space data is maintained in a series of planes which can be accessed by various methods within this class.

The text plane contains the actual characters in the presentation space. Most of the methods in OhioScreen class work exclusively with the text plane.

The remaining planes contain the corresponding attributes for each character in the text plane. The color plane contains color characteristics. The field plane contains the field attributes.

The extended plane contains the extended field attributes. The color, field, and extended planes are not interpreted by any of the methods in this class.

See Also OhioScreen Properties, Methods and Events.

5.2.2.5.1 Methods

- GetData
- FindString
- Refresh
- SendAid
- SendKeys
- SetString
- Wait
- WaitFor
5.2.2.5.1.1 GetData

Returns a character array containing the data from the Text, Color, Field or Extended plane of the virtual screen.

Visual Basic Syntax

[Char Array =] Screen.GetData(start As TTnbOhioPosition, end As TTnbOhioPosition, plane As Integer)

Delphi Syntax

[TCharArr :=] Screen.GetData(startpos, endpos:TTnbOhioPosition; plane:integer);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>The row and column where to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the data). &quot;start&quot; must be positionally less than &quot;end&quot;.</td>
</tr>
<tr>
<td>end</td>
<td>The row and column where to end. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the ending location and 1,1 will be included in the data). &quot;end&quot; must be positionally greater than &quot;start&quot;.</td>
</tr>
<tr>
<td>plane</td>
<td>A valid OHIO_PLANE value.</td>
</tr>
</tbody>
</table>

5.2.2.5.1.2 FindString

Searches the text plane for the target string. If found, returns an OhioPosition object containing the target location. If not found, returns a null. The TargetString must be completely contained by the target area for the search to be successful. Null characters in the text plane are treated as blank spaces during search processing.

Visual Basic Syntax

[TTnbOhioPosition =] Screen.FindString(TargetString As String, startPos As
OhioPosition, length As Integer, dir As OHIO_DIRECTION, IgnoreCase As Boolean)

Delphi Syntax

[TTnbOhioPosition :=] Screen.FindString(TargetString:string;
startPos:TTnbOhioPosition; length:integer, dir:integer; IgnoreCase:boolean);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetString</td>
<td>The target string.</td>
</tr>
<tr>
<td>startPos</td>
<td>The row and column where to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the search).</td>
</tr>
<tr>
<td>length</td>
<td>The length from startPos to include in the search.</td>
</tr>
<tr>
<td>dir</td>
<td>An OHIO_DIRECTION value.</td>
</tr>
<tr>
<td>IgnoreCase</td>
<td>Indicates whether the search is case sensitive. True means that case will be ignored. False means the search will be case sensitive.</td>
</tr>
</tbody>
</table>

5.2.2.5.1.3 Refresh

Updates the collection of OhioScreen objects. All OhioScreen objects that are available on the system at the time of the refresh will be added to the collection. Indexing of OhioScreen objects will not be preserved across refreshes.

Visual Basic Syntax

Screen.Refresh

Delphi Syntax

Screen.Refresh

Parameters

None.
5.2.2.5.1.4 SendAid

The SendAid method sends an "aid" keystroke to the virtual screen. These aid keys can be thought of as special keystrokes, like the Enter key, the Tab key, or the Page Up key. All the valid special key values are contained in the OHIO_AID enumeration.

**Visual Basic Syntax**

```
Screen.SendAid(aidkey As Integer)
```

**Delphi Syntax**

```
Screen.SendAid(aidKey:integer);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aidKey</td>
<td>The aid key to send to the virtual screen.</td>
</tr>
</tbody>
</table>

5.2.2.5.1.5 SendKeys

The SendKeys method sends a string of keys to the virtual screen. This method acts as if keystrokes were being typed from the keyboard. The keystrokes will be sent to the location given. If no location is provided, the keystrokes will be sent to the current cursor location.

**Visual Basic Syntax**

```
Screen.SendKeys(text As String, location As OhioPosition)
```

**Delphi Syntax**

```
Screen.SendKeys(text:string; location:TTnbOhioPosition);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>The string of characters to be sent.</td>
</tr>
<tr>
<td>location</td>
<td>OhioPosition where the string should be written.</td>
</tr>
</tbody>
</table>
5.2.2.5.1.6 SetString

The SetString method sends a string to the virtual screen at the specified location. The string will overlay only unprotected fields, and any parts of the string which fall over protected fields will be discarded.

**Visual Basic Syntax**

```vbnet
Screen.SetString(text As String, location As TTnbOhioPosition)
```

**Delphi Syntax**

```delphi
Screen.SetString(text:string; location:TTnbOhioPosition)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>String to place in the virtual screen</td>
</tr>
<tr>
<td>location</td>
<td>Position where the string should be written</td>
</tr>
</tbody>
</table>

5.2.2.5.1.7 Wait

General wait mechanism. This method basically waits until the mainframe application turns in an unlocked state and is able to receive input data.

**Visual Basic Syntax**

```vbnet
[Boolean] = Screen.Wait([TimeOut] As Integer)
```

**Delphi Syntax**

```delphi
[WordBool] = Screen.Wait([TimeOut]:Integer)
```

**Remarks**

The Timeout parameter is optional and it is measured in milliseconds. This method returns True if the wait was successful and False if the timeout period has expired.

See also [WaitForConnect](#), [WaitForUnlock](#) and [WaitForNewScreen](#) methods.
5.2.2.5.1.8 WaitFor

This method waits for a screen containing the specified string. In the first case shown below, it returns True if the wait was successful and False if the Timeout period has expired. In the second case it receives an array of strings as parameter, and returns an integer indicating the number of the string (into the array of strings) that was found.

**Visual Basic Syntax**

```vbnet
[Boolean =] Screen.WaitFor(Value As String, [TimeOut] As Integer, ResetWait As Boolean)

[Integer =] Screen.WaitFor(Values As String, [TimeOut] As Integer, ResetWait As Boolean)
```

**Delphi Syntax**

```delphi
[WordBool :=] Screen.WaitFor(Value:string;[TimeOut]:Integer; ResetWait:Boolean);

[Integer :=] Screen.WaitFor(const Values:Array of string;[TimeOut]:Integer; ResetWait:Boolean);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>The string of characters expected to be found.</td>
</tr>
<tr>
<td>Values</td>
<td>The array of strings where one of them is expected to be found. In Visual Basic this is a string with several values separated by &quot;;&quot;</td>
</tr>
<tr>
<td>TimeOut</td>
<td>Optional parameter measured in milliseconds.</td>
</tr>
<tr>
<td>ResetWait</td>
<td>While this parameter is false, continue lopping until the expected string is found.</td>
</tr>
</tbody>
</table>

5.2.2.5.1.9 WaitForNewScreen

This method waits until a new screen arrives within the specified period of time. It is similar to WaitForScreen method except that it always waits for a new screen arrival.

This method returns True if the wait was successful and False if the Timeout period has expired.

**Visual Basic Syntax**
5.2.2.5.1.10  WaitForNewUnlock

This method waits until a new system unlock arrives within the specified period of time. It is similar to WaitForUnlock method except that it always waits for a new system unlock arrival. It returns True if the wait was successful and False if the Timeout period has expired.

**Visual Basic Syntax**

[Boolean =] Screen.WaitForNewUnlock([TimeOut] As Integer)

**Delphi Syntax**

[Boolean :=] Screen.WaitForNewUnlock([TimeOut]: Integer);

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOut</td>
<td>Optional parameter measured in milliseconds.</td>
</tr>
</tbody>
</table>

5.2.2.5.1.11  WaitForScreen

This method waits for the first screen after a system lock. If the host system changes from an unlocked to a locked state (normally when we send an Aid key) the method waits until the first screen arrives. If WaitForScreen method is called after that event, it returns immediately. The method returns True if the wait was successful and False if the Timeout period has expired.

**Visual Basic Syntax**

[Boolean =] Screen.WaitForScreen([TimeOut] As Integer)

**Delphi Syntax**

[Boolean :=] Screen.WaitForScreen([TimeOut]: Integer);

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeOut</td>
<td>Optional parameter measured in milliseconds.</td>
</tr>
</tbody>
</table>
5.2.2.5.1.12 WaitForUnlock

This method waits until a new system unlock arrives within the specified period of time. It is similar to WaitForUnlock method except that it always waits for a new system unlock arrival. This method returns True if the wait was successful and False if the Timeout period has expired.

Visual Basic Syntax

[Boolean =] Screen.WaitForUnlock([Timeout] As Integer)

Delphi Syntax

[WordBool :=] Screen.WaitForUnlock([Timeout]: Integer);

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout</td>
<td>Optional parameter measured in milliseconds.</td>
</tr>
</tbody>
</table>

5.2.2.5.2 Properties

- Columns
- Cursor
- Fields
5.2.2.5.2.1 Columns

Indicates the number of columns in the presentation space.

**Visual Basic Syntax**

 `[Integer =] Screen.Columns`

**Delphi Syntax**

 `[Integer :=] Screen.Columns;`

**Remarks**

This property returns an integer representing the number of columns on the current host screen.

5.2.2.5.2.2 Cursor

Specifies the location of the cursor in the presentation space. The row and column of the cursor is contained within the OhioPosition object.

**Visual Basic Syntax**

 `Screen.Cursor [= TTnbOhioPosition]`

**Delphi Syntax**

 `Screen.Cursor; [:= TTnbOhioPosition]`

**Remarks**

This function returns an integer which represents the current cursor position, or sets the cursor position according to the OhioPosition passed as parameter.
5.2.2.5.2.3 Fields

Returns the OhioFields object associated with this presentation space. This provides another way to access the data in the virtual screen. The OhioFields object contains a snapshot of all the fields in the current virtual screen. Fields provide methods for interpreting the data in the non-text planes. Zero length fields (due to adjacent field attributes) are not returned in the OhioFields collection. For unformatted screens, the returned collection contains only one OhioField that contains the whole virtual screen.

Visual Basic Syntax

```vbnet
[TTnbOhioFields =] Screen.Fields
```

Delphi Syntax

```delphi
[TTnbOhioFields :=] Screen.Fields;
```

Remarks

Returns the Fields object for the current host screen.

5.2.2.5.2.4 OIA

Returns the OIA object associated with this presentation space.

Visual Basic Syntax

```vbnet
[TTnbOhioOIA =] Screen.OIA
```

Delphi Syntax

```delphi
[TTnbOhioOIA :=] Screen.OIA;
```

Remarks

This object can be used to query the status of the operator information area.
5.2.2.5.2.5 Rows

Indicates the number of rows in the presentation space.

**Visual Basic Syntax**

```vbnet
[Integer =] Screen.Rows
```

**Delphi Syntax**

```delphi
[Integer :=] Screen.Rows;
```

**Remarks**

This property returns an integer representing the number of rows on the current host screen.

5.2.2.5.2.6 String/Text

Returns the entire text plane of the virtual screen as a string.

**Visual Basic Syntax**

```vbnet
[String =] Screen.String
```

**Delphi Syntax**

```delphi
[String :=] Screen.Text;
```

**Remarks**

All null characters and Field Attribute characters are returned as blank space characters.

In Delphi `String` is used as `Text` property because the name `String` can't be used as a function name.

5.2.2.5.3 Events

- **OnConnect**
- **OnConnectionFail**
- **OnDisconnect**
- **OnScreenChange**
5.2.2.5.3.1 OnConnect

Occurs after a successfully connection to the server is established.

5.2.2.5.3.2 OnConnectionFail

Occurs after the connection to the server fails.

5.2.2.5.3.3 OnDisconnect

Occurs after a disconnection of the server.

5.2.2.5.3.4 OnScreenChange

This event is generated whenever the virtual screen is modified.

5.2.2.5.3.5 OnSendAid

Occurs before an Aid key is to be sent.

Remarks

This event is fired when a SendAid or SendKeys methods are used to send an Aid Key and data to the mainframe. Occurs before the data is sent, allowing to take actions like filling edit fields or saving data typed in the emulator control.
5.2.2.5.3.6 OnSessionState

Occurs when a session-state change takes place.

Remarks

A session state can be ssNoSession, ssSSCPLU and ssLULU. Any transition between the states fires this event.

5.2.2.6 OhioFields

OhioFields contains a collection of the fields in the virtual screen. It provides methods to iterate through the fields, find fields based on location, and find fields containing a given string. Each element of the collection is an instance of OhioField. OhioFields can only be accessed through OhioScreen using the Fields property. OhioFields is a static view of the virtual screen and does not reflect changes made to the virtual screen after its construction. The field list can be updated with a new view of the virtual screen using the Refresh() method.

Note: All OhioFields objects returned by methods in this class are invalidated when Refresh() is called.

See also OhioFields Properties and Methods.

5.2.2.6.1 Methods

- Item
- FindByString
- FindByPosition
- Refresh

5.2.2.6.1.1 Item

Indicates the OhioSession object at the given index. "One based" indexing is used in all Ohio collections. For example, the first OhioSession in this collection is at index 1.
Visual Basic Syntax

[TTnbOhioField =] Fields.Item(index As Integer)

Delphi Syntax

[TTnbOhioField :=] Fields.Item(index:integer);

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>The index of the target OhioSession.</td>
</tr>
</tbody>
</table>

Returns null if no object with that name exists in this collection.

5.2.2.6.1.2 FindByPosition

Searches the collection for the target position and returns the OhioField object containing that position. If not found, returns a null.

Visual Basic Syntax

[TTnbOhioField =] FieldsFindByPosition(targetPosition As TTnbOhioPosition)

Delphi Syntax

[TTnbOhioField :=] FieldsFindByPosition(targetString:TTnbOhioPosition);

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetPosition</td>
<td>The target row and column.</td>
</tr>
</tbody>
</table>

5.2.2.6.1.3 FindByString

Searches the collection for the target string and returns the OhioField object containing that string. The string must be totally contained within the field to be considered a match. If the target string is not found, a null will be returned.
Visual Basic Syntax

\[TTnbOhioField = \] Fields.Fprintf.string(targetString As String, startPos As TTnbOhioPosition, length As Integer, dir As Integer, ignoreCase As Boolean)

Delphi Syntax

\[TTnbOhioField :=\] Fields.FindByString(targetString:string; startPos:TTnbOhioPosition; length:integer; dir:integer; ignoreCase:boolean);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetString</td>
<td>The target string.</td>
</tr>
<tr>
<td>startPos</td>
<td>The row and column where to start. The position is inclusive (for example, row 1, col 1 means that position 1,1 will be used as the starting location and 1,1 will be included in the search).</td>
</tr>
<tr>
<td>length</td>
<td>The length from startPos to include in the search.</td>
</tr>
<tr>
<td>dir</td>
<td>An OHIO_DIRECTION value.</td>
</tr>
<tr>
<td>ignoreCase</td>
<td>Indicates whether the search is case sensitive. True means that case will be ignored. False means the search will be case sensitive.</td>
</tr>
</tbody>
</table>

5.2.2.6.1.4 Refresh

Updates the collection of OhioSession objects. All OhioSession objects that are available on the system at the time of the refresh will be added to the collection. Indexing of OhioSession objects will not be preserved across refreshes.

Visual Basic Syntax

Fields.Refresh

Delphi Syntax

Fields.Refresh;

Parameters

None.
5.2.2.6.2 Properties

- **Columns**
- **Count**
- **Rows**

5.2.2.6.2.1 Columns

Indicates the number of columns in the presentation space.

**Visual Basic Syntax**

\[
\text{[Integer =]} \ Fields.\text{Columns}
\]

**Delphi Syntax**

\[
\text{[Integer :=]} \ Fields.\text{Columns;}
\]

**Remarks**

This property returns an integer representing the number of columns on the current host screen.

5.2.2.6.2.2 Count

Returns the number of OhioSession objects contained in this collection.

**Visual Basic Syntax**

\[
\text{[Integer =]} \ Fields.\text{Count}
\]

**Delphi Syntax**

\[
\text{[Integer :=]} \ Fields.\text{Count;}
\]
5.2.2.6.2.3 Rows

Indicates the number of rows in the presentation space.

**Visual Basic Syntax**

`[Integer =] Fields.Rows`

**Delphi Syntax**

`[Integer :=] Fields.Rows;`

**Remarks**

This property returns an integer representing the number of rows on the current host screen.

5.2.2.7 OhioField

A field is the fundamental element of a virtual screen. A field includes both data and attributes describing the field. The OhioField class encapsulates a virtual screen field and provides methods for accessing and manipulating field attributes and data. OhioField objects can be accessed only through the OhioFields object.

See also **Properties** and **Methods**.

5.2.2.7.1 Methods

- **GetData**

5.2.2.7.1.1 GetData

Returns a character array containing the data from the Text, Color, Field or Extended plane of the virtual screen.

**Visual Basic Syntax**
[Array =] Field.GetData(plane As Integer)

Delphi Syntax

[TCharArr :=] Field.GetData(plane:integer);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plane</td>
<td>A valid OHIO_PLANE value.</td>
</tr>
</tbody>
</table>

5.2.2.7.2 Properties

- Attribute
- EndPos
- Hidden
- HighIntensity
- Length
- Modified
- Numeric
- PenSelectable
- ProtectedField
- StartPos
- String/Text

5.2.2.7.2.1 Attribute

Indicates the attribute byte for the field.

Visual Basic Syntax

[Integer =] Field.Attribute

Delphi Syntax

[Integer :=] Field.Attribute;
5.2.2.7.2.2 EndPos

Returns the ending position of the field. The position can range from 1 to the size of the virtual screen. The ending position of a field is the position of the last character in the field.

**Visual Basic Syntax**

```vbnet
[TTrnOhioPosition =] Field.EndPos
```

**Delphi Syntax**

```delphi
[TTrnOhioPosition :=] Field.EndPos;
```

**Remarks**

Returns an integer indicating the position of the last character in the field.

5.2.2.7.2.3 Hidden

Indicates whether or not the field is hidden.

**Visual Basic Syntax**

```vbnet
[Boolean =] Field.Hidden
```

**Delphi Syntax**

```delphi
```

**Remarks**

This property returns True if the field is hidden, otherwise false.

5.2.2.7.2.4 HighIntensity

Indicates whether or not the field is high-intensity.

**Visual Basic Syntax**

```vbnet
[Boolean =] Field.HighIntensity
```
Delphi Syntax

[Boolean :=] Field.HighIntensity;

Remarks

Returns True if the field is high intensity, otherwise False.

5.2.2.7.2.5 Length

Returns the length of the field.

Visual Basic Syntax

[Integer =] Field.Length

Delphi Syntax

[Integer :=] Field.Length;

Remarks

A field's length can range from 1 to the size of the virtual screen.

5.2.2.7.2.6 Modified

Indicates whether or not the field has been modified.

Visual Basic Syntax

[Boolean =] Field.Modified

Delphi Syntax

[Boolean :=] Field.Modified;

Remarks

This property returns True if the field has been modified, otherwise False.
5.2.2.7.2.7 Numeric
Indicates whether the field which contains the cursor is a numeric-only field.

**Visual Basic Syntax**

[Boolean =] Field.Numeric

**Delphi Syntax**


**Remarks**

This property returns True if the cursor is in a numeric-only field, false otherwise.

5.2.2.7.2.8 PenSelectable
Indicates whether or not the field is pen-selectable.

**Visual Basic Syntax**

[Boolean =] Field.PenSelectable

**Delphi Syntax**


**Remarks**

This property returns True if the field is pen-selectable, otherwise False.

5.2.2.7.2.9 ProtectedField
Indicates whether or not the field is protected.

**Visual Basic Syntax**
[Boolean =] Field.Protected

**Delphi Syntax**

[Boolean :=] Field.Protected;

**Remarks**

This property returns True if the field is protected, otherwise False.

---

**5.2.2.7.2.10 StartPos**

Indicates the starting position of the field.

**Visual Basic Syntax**

[TTnbOhioPosition =] Field.StartPos

**Delphi Syntax**

[TTnbOhioPosition :=] Field.StartPos;

**Remarks**

The position can range from 1 to the size of the virtual screen. The starting position of a field is the position of the first character in the field.

---

**5.2.2.7.2.11 String/Text**

Returns the entire text plane of the virtual screen as a string. String and Text are both the same properties, but String can't be used in Delphi as a function name.

**Visual Basic Syntax**

Field.String [= String]

**Delphi Syntax**

Field.Text; [= String]
5.2.2.8 OhioOIA

The operator information area of a host session. This area is used to provide status information regarding the state of the host session and location of the cursor. An OhioOIA object can be obtained using the OIA() method on an instance of OhioScreen.

5.2.2.8.1 Properties

- **Alphanumeric**
- **CommCheckCode**
- **InputInhibited**
- **MachineCheckCode**
- **Numeric**
- **Owner**
- **ProgCheckCode**

5.2.2.8.1.1 AlphaNumeric

Indicates whether the field which contains the cursor is an alphanumeric field. True if the cursor is in an alphanumeric field, false otherwise.

**Visual Basic Syntax**

```vbnet
[Boolean=] OIA.Alphanumeric
```

**Delphi Syntax**

```delphi
[Boolean :=] OIA.Alphanumeric;
```

**Remarks**

This property returns True if the cursor is in an alphanumeric field, False in the other
5.2.2.8.1.2 CommCheckCode

Returns the communication check code.

Visual Basic Syntax

[Integer =] OIA.CommCheckCode

Delphi Syntax

[Integer :=] OIA.CommCheckCode;

Remarks
If InputInhibited returns OHIO_INPUTINHIBITED_COMMCHECK, this property will return the communication check code.

5.2.2.8.1.3 InputInhibited

Indicates whether or not input is inhibited.

Visual Basic Syntax

[Integer =] OIA.InputInhibited

Delphi Syntax

[Integer :=] OIA.InputInhibited;

Remarks
If input is inhibited, SendKeys or SendAID calls to the OhioScreen are not allowed. Why input is inhibited can be determined from the value returned. If input is inhibited for more than one reason, the highest value is returned.
5.2.2.8.1.4 MachineCheckCode

Specifies the machine check code.

**Visual Basic Syntax**

\[\text{Integer } = \text{ OIA.MachineCheckCode}\]

**Delphi Syntax**

\[\text{Integer } := \text{ OIA.MachineCheckCode};\]

**Remarks**

If InputInhibited returns OHIO_INPUTINHIBITED_MACHINECHECK, this property will return the machine check code.

5.2.2.8.1.5 Numeric

Indicates whether the field which contains the cursor is a numeric-only field.

**Visual Basic Syntax**

\[\text{Boolean } = \text{ OIA.Numeric}\]

**Delphi Syntax**

\[\text{Boolean } := \text{ OIA.Numeric};\]

**Remarks**

This property returns True if the cursor is in a numeric-only field, false otherwise.

5.2.2.8.1.6 Owner

Indicates the owner of the host connection.

**Visual Basic Syntax**

\[\text{Integer } = \text{ OIA.Owner}\]

**Delphi Syntax**
[Integer :=] OIA.Owner;

Remarks
The Integer returned by this property specifies one of the OHIO_OWNER values.

5.2.2.8.1.7 ProgCheckCode
Returns the program check code.

Visual Basic Syntax
[Integer =] OIA.ProgCheckCode

Delphi Syntax
[Integer :=] OIA.ProgCheckCode;

Remarks
If InputInhibited returns OHIO_INPUTINHIBITED_PROGCHECK, this property will return the program check code.

5.2.2.9 OhioPosition
Holds row and column coordinates. An OhioPosition can be constructed by using CreateOhioPosition() on any Ohio class.

See also OhioPosition Properties and Methods.

5.2.2.9.1 Methods

- Refresh
5.2.2.9.1.1 Refresh

Updates the collection of OhioSession objects. All OhioSession objects that are available on the system at the time of the refresh will be added to the collection. Indexing of OhioSession objects will not be preserved across refreshes.

**Visual Basic Syntax**

```
Position.Refresh(TnbFields As TTnbFields, fromPos As Integer)
```

**Delphi Syntax**

```
Position.Refresh(TnbFields:TTnbFields; fromPos:integer);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TnbFields</td>
<td>The fields to refresh.</td>
</tr>
<tr>
<td>fromPos</td>
<td>The position where to start.</td>
</tr>
</tbody>
</table>

5.2.2.9.2 Properties

- **Col**
- **Pos**
- **Row**

5.2.2.9.2.1 Col

Indicates the column coordinate.

**Visual Basic Syntax**

```
Position.Col [= Integer]
```

**Delphi Syntax**

```
Position.Col; [:= Integer]
```
5.2.2.9.2.2 Pos

Gets or sets the current position.

**Visual Basic Syntax**

`Position.Pos [= Integer]`

**Delphi Syntax**

`Position.Pos; [= Integer]`

**Remarks**

Value indicates the position to set to the `Position` object.

5.2.2.9.2.3 Row

Indicates the row coordinate.

**Visual Basic Syntax**

`Position.Row [= Integer]`

**Delphi Syntax**

`Position.Row; [= Integer]`

5.2.3 Constants

- [Ohio Constants](#)
- [3270 Key Values](#)
- [5250 Key Values](#)
5.2.3.1 Ohio Constants

Constants for all Ohio classes.

**OHIO_INFO**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_VERSION_LEVEL</td>
<td>OHIO 1.00</td>
<td>The OHIO version level of this implementation. The form is “OHIO nn.nn”</td>
</tr>
<tr>
<td>OHIO_VENDOR_NAME</td>
<td>Cybele Software Inc.</td>
<td>The name of the vendor providing this OHIO implementation. Format is vendor defined.</td>
</tr>
<tr>
<td>OHIO_VENDOR_PRODUCT_VERSION</td>
<td>TN Bridge Integration Pack 3.0</td>
<td>The vendor product version that is providing the OHIO implementation. Format is vendor specific.</td>
</tr>
</tbody>
</table>

**uTnbOHIO_SESSION cfg keys**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_SESSION_TYPE</td>
<td>OHIO_SESSION_TYPE=</td>
<td>The OHIO Session Type: unknown host, 3270 or 5250 host.</td>
</tr>
<tr>
<td>OHIO_SESSION_HOST</td>
<td>OHIO_SESSION_HOST=</td>
<td>The OHIO Session Host.</td>
</tr>
<tr>
<td>OHIO_SESSION_PORT</td>
<td>OHIO_SESSION_PORT=</td>
<td>The OHIO Session Port.</td>
</tr>
<tr>
<td>OHIO_SESSION_NAME</td>
<td>OHIO_SESSION_NAME=</td>
<td>The OHIO Session Name.</td>
</tr>
<tr>
<td>OHIO_SESSION_CONFIGRESOURCE</td>
<td>OHIO_SESSION_CONFIGRESOURCE=</td>
<td>The OHIO Session Configuration Resource.</td>
</tr>
<tr>
<td>OHIO_SESSION_TN_ENHANCED</td>
<td>OHIO_SESSION_TN_ENHANCED =</td>
<td></td>
</tr>
<tr>
<td>OHIO_SESSION_SCREEN_SIZE</td>
<td>OHIO_SESSION_SCREEN_SIZE=</td>
<td>The Screen Size for the OHIO Session.</td>
</tr>
<tr>
<td>OHIO_SESSION_CODE_PAGE</td>
<td>OHIO_SESSION_CODE_PAGE=</td>
<td>The Code Page for the OHIO Session.</td>
</tr>
</tbody>
</table>

**OHIO_DIRECTION**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_DIRECTION_FORWARD</td>
<td>0</td>
<td>Forward (beginning towards end)</td>
</tr>
<tr>
<td>OHIO_DIRECTION_BACKWARD</td>
<td>1</td>
<td>Backward (end towards beginning)</td>
</tr>
</tbody>
</table>

**OHIO_TYPE**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_TYPE_UNKNOWN</td>
<td>0</td>
<td>Unknown Host</td>
</tr>
<tr>
<td>OHIO_TYPE_3270</td>
<td>1</td>
<td>3270 Host</td>
</tr>
</tbody>
</table>
### OHIO_TYPE_5250

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5250 Host</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### OHIO_STATE

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_STATE_DISCONNECTED</td>
<td>0</td>
<td>The communication link to the host is disconnected</td>
</tr>
<tr>
<td>OHIO_STATE_CONNECTED</td>
<td>1</td>
<td>The communication link to the host is connected</td>
</tr>
</tbody>
</table>

### OHIO_PLANE

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_PLANE_TEXT</td>
<td>1</td>
<td>Indicates Plane Text (character data)</td>
</tr>
<tr>
<td>OHIO_PLANE_COLOR</td>
<td>2</td>
<td>Indicates Plane Color (standard HLLAPI CGA color values)</td>
</tr>
<tr>
<td>OHIO_PLANE_FIELD</td>
<td>4</td>
<td>Indicates Field Attribute Plane (field attribute bytes)</td>
</tr>
<tr>
<td>OHIO_PLANE_EXTENDED</td>
<td>8</td>
<td>Indicates Extended Plane (extended attribute bytes)</td>
</tr>
</tbody>
</table>

### OHIO_COLOR

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_COLOR_BLACK</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>OHIO_COLOR_BLUE</td>
<td>1</td>
<td>Blue</td>
</tr>
<tr>
<td>OHIO_COLOR_GREEN</td>
<td>2</td>
<td>Green</td>
</tr>
<tr>
<td>OHIO_COLOR_CYAN</td>
<td>3</td>
<td>Cyan</td>
</tr>
<tr>
<td>OHIO_COLOR_RED</td>
<td>4</td>
<td>Red</td>
</tr>
<tr>
<td>OHIO_COLOR_MAGENTA</td>
<td>5</td>
<td>Magenta</td>
</tr>
<tr>
<td>OHIO_COLOR_YELLOW</td>
<td>6</td>
<td>Yellow</td>
</tr>
<tr>
<td>OHIO_COLOR_WHITE</td>
<td>7</td>
<td>White</td>
</tr>
</tbody>
</table>

### OHIO_EXTENDED

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_EXTENDED_HILITE</td>
<td>$C0</td>
<td>Bitmask for Highlighting Bits</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR</td>
<td>$38</td>
<td>Bitmask for Color Bits</td>
</tr>
<tr>
<td>OHIO_EXTENDED_RESERVED</td>
<td>$07</td>
<td>Bitmask for Reserved Bits</td>
</tr>
<tr>
<td>OHIO_EXTENDED_HILITE_NORMAL</td>
<td>$00</td>
<td>Normal highlighting</td>
</tr>
<tr>
<td>OHIO_EXTENDED_HILITE_BLINK</td>
<td>$40</td>
<td>Blinking highlighting</td>
</tr>
<tr>
<td>OHIO_EXTENDED_HILITE_REVERSE_VIDEO</td>
<td>$80</td>
<td>Reverse video highlighting</td>
</tr>
<tr>
<td>OHIO_EXTENDED_HILITE_UNDERSCORE</td>
<td>$C0</td>
<td>Under score highlighting</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_DEFAULT</td>
<td>$00</td>
<td>Default color</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_BLUE</td>
<td>$08</td>
<td>Blue</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_RED</td>
<td>$10</td>
<td>Red</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_PINK</td>
<td>$18</td>
<td>Pink</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_GREEN</td>
<td>$20</td>
<td>Green</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_TURQUOISE</td>
<td>$28</td>
<td>Turquoise</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_YELLOW</td>
<td>$30</td>
<td>Yellow</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLOR_WHITE</td>
<td>$38</td>
<td>White</td>
</tr>
<tr>
<td>OHIO_EXTENDED_COLUMN_SEPARATOR</td>
<td>$02</td>
<td>Separator</td>
</tr>
</tbody>
</table>

**OHIO_FIELD**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_FIELD_ATTRIBUTE</td>
<td>$C0</td>
<td>Bitmask for field attribute</td>
</tr>
<tr>
<td>OHIO_FIELD_PROTECTED</td>
<td>$20</td>
<td>Protected field</td>
</tr>
<tr>
<td>OHIO_FIELD_NUMERIC</td>
<td>$10</td>
<td>Numeric field</td>
</tr>
<tr>
<td>OHIO_FIELD_HIDDEN</td>
<td>$0C</td>
<td>Hidden field</td>
</tr>
<tr>
<td>OHIO_FIELD_PEN_SELECTABLE</td>
<td>$08</td>
<td>Pen selectable field</td>
</tr>
<tr>
<td>OHIO_FIELD_HIGH_INTENSITY</td>
<td>$04</td>
<td>High Intensity field</td>
</tr>
<tr>
<td>OHIO_FIELD_RESERVED</td>
<td>$02</td>
<td>Reserved field</td>
</tr>
<tr>
<td>OHIO_FIELD_MODIFIED</td>
<td>$01</td>
<td>Modified field</td>
</tr>
</tbody>
</table>

**OHIO_UPDATE**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_UPDATE_CLIENT</td>
<td>0</td>
<td>Update initiated by client</td>
</tr>
<tr>
<td>OHIO_UPDATE_HOST</td>
<td>1</td>
<td>Update initiated by host</td>
</tr>
</tbody>
</table>

**OHIO_OWNER**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_OWNER_UNKNOWN</td>
<td>0</td>
<td>Uninitialized</td>
</tr>
<tr>
<td>OHIO_OWNER_APP</td>
<td>1</td>
<td>Application or 5250 host</td>
</tr>
<tr>
<td>OHIO_OWNER_MYJOB</td>
<td>2</td>
<td>3270 - Myjob</td>
</tr>
<tr>
<td>OHIO_OWNER_NVT</td>
<td>3</td>
<td>NVT (Network Virtual Terminal) mode</td>
</tr>
<tr>
<td>OHIO_OWNER_UNOWNED</td>
<td>4</td>
<td>Unowned (3270 only)</td>
</tr>
<tr>
<td>OHIO_OWNER_SSCP</td>
<td>5</td>
<td>SSCP (3270 only)</td>
</tr>
</tbody>
</table>

**OHIO_INPUTINHIBITED**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_INPUTINHIBITED_NOTINHIBITED</td>
<td>0</td>
<td>Input not inhibited</td>
</tr>
<tr>
<td>OHIO_INPUTINHIBITED_SYSTEM_WAIT</td>
<td>1</td>
<td>Input inhibited by a System Wait state (&quot;X SYSTEM&quot; or &quot;X [&quot;]&quot;)</td>
</tr>
<tr>
<td>OHIO_INPUTINHIBITED_COMMCHECK</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
5.2.3.2 3270 Key Values

<table>
<thead>
<tr>
<th>Aid Key Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_AID_KEY_ATTN</td>
<td>$0182</td>
</tr>
<tr>
<td>OHIO_AID_KEY_BACKSPACE</td>
<td>$0183</td>
</tr>
<tr>
<td>OHIO_AID_KEY_BACKTAB</td>
<td>$0184</td>
</tr>
<tr>
<td>OHIO_AID_KEY_CLEAR</td>
<td>$0186</td>
</tr>
<tr>
<td>OHIO_AID_KEY_CURSORSEL</td>
<td>$0188</td>
</tr>
<tr>
<td>OHIO_AID_KEY_DELETE</td>
<td>$0189</td>
</tr>
<tr>
<td>OHIO_AID_KEY_DOWN</td>
<td>$018B</td>
</tr>
<tr>
<td>OHIO_AID_KEY_DUP</td>
<td>$018C</td>
</tr>
<tr>
<td>OHIO_AID_KEY_ENTER</td>
<td>$018D</td>
</tr>
<tr>
<td>OHIO_AID_KEY_ERASEEOF</td>
<td>$018E</td>
</tr>
<tr>
<td>OHIO_AID_KEY_ERASEINPUT</td>
<td>$018F</td>
</tr>
<tr>
<td>OHIO_AID_KEY_HOME</td>
<td>$0190</td>
</tr>
</tbody>
</table>
### 5250 Key Values

<table>
<thead>
<tr>
<th>Aid Key Constants</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OHIO_AID_KEY_ATTN</td>
<td>$0182</td>
</tr>
<tr>
<td>OHIO_AID_KEY_BACKSPACE</td>
<td>$0183</td>
</tr>
<tr>
<td>OHIO_AID_KEY_BACKTAB</td>
<td>$0184</td>
</tr>
<tr>
<td>OHIO_AID_KEY_CLEAR</td>
<td>$0186</td>
</tr>
<tr>
<td>OHIO_AID_KEY_CURSORSEL</td>
<td>$0188</td>
</tr>
<tr>
<td>OHIO_AID_KEY_LEFT</td>
<td>$0193</td>
</tr>
<tr>
<td>OHIO_AID_KEY_MARK</td>
<td>$0194</td>
</tr>
<tr>
<td>OHIO_AID_KEY_NEWLINE</td>
<td>$0195</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PA1</td>
<td>$0196</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PA2</td>
<td>$0197</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PA3</td>
<td>$0198</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF1</td>
<td>$01A0</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF2</td>
<td>$01A1</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF3</td>
<td>$01A2</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF4</td>
<td>$01A3</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF5</td>
<td>$01A4</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF6</td>
<td>$01A5</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF7</td>
<td>$01A6</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF8</td>
<td>$01A7</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF9</td>
<td>$01A8</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF10</td>
<td>$01A9</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF11</td>
<td>$01AA</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF12</td>
<td>$01AB</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF13</td>
<td>$01AC</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF14</td>
<td>$01AD</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF15</td>
<td>$01AE</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF16</td>
<td>$01AF</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF17</td>
<td>$01B0</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF18</td>
<td>$01B1</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF19</td>
<td>$01B2</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF20</td>
<td>$01B3</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF21</td>
<td>$01B4</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF22</td>
<td>$01B5</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF23</td>
<td>$01B6</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PF24</td>
<td>$01B7</td>
</tr>
<tr>
<td>OHIO_AID_KEY_RESET</td>
<td>$019A</td>
</tr>
<tr>
<td>OHIO_AID_KEY_RIGHT</td>
<td>$019B</td>
</tr>
<tr>
<td>OHIO_AID_KEY_SYSREQ</td>
<td>$019C</td>
</tr>
<tr>
<td>OHIO_AID_KEY_TABFORWARD</td>
<td>$019D</td>
</tr>
<tr>
<td>OHIO_AID_KEY_UP</td>
<td>$019F</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PRINT</td>
<td>$01C0</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PGUP</td>
<td>$01C1</td>
</tr>
<tr>
<td>OHIO_AID_KEY_PGDOWN</td>
<td>$01C2</td>
</tr>
</tbody>
</table>
6 XML Reference

- **Introduction**
- **Programming Reference**

### 6.1 Introduction

XML (eXtensible Markup Language) is a framework for defining markup languages:

- There is no fixed collection of markup tags - you may define your own tags, tailored for your kind of information.
- Each XML language is targeted at its own application domain, but the languages will share many features.
- There is a common set of generic tools for processing documents.

XML is designed to:
• Separate syntax from semantics to provide a common framework for structuring information (browser rendering semantics is completely defined by stylesheets).
• Allow tailor-made markup for any imaginable application domain.
• Support internationalization (Unicode) and platform independence.
• Be the future of structured information, including databases.

6.2 Programming Reference

Just like in previous versions of TN Bridge, when working with profiles, you can operate with a single or several files. Main difference in this version is that subkeys are fixed. When you set the StreamingType property of the TNBProfiles component to XML value, subkeys will be predetermined tags, and the Key will be `TNBRIDGE` tag, that is the root.

The following is the tags' hierarchy we will use:

```xml
<TNBRIDGE>
  <Connections>
    <Connection>
      <Host ... />
      <Display ... />
    </Connection>
    ...
    ...
  </Connections>
  <ScreenStyles>
    <ScreenStyle>
      <General>
        <Font ... />
        <General />
      </General>
      <FieldMapping>
        <FieldMap>
          <Input ... />
          <Output ... />
        </FieldMap>
        ...
      </FieldMapping>
    </ScreenStyle>
    ...
    ...
    <ScreenStyles />
  </ScreenStyles>
  <KeyboardMaps>
    <KeyboardMap>
      <KeyMap>
        <FunctionKey ... />
      </KeyMap>
    </KeyboardMap>
  </KeyboardMaps>
</TNBRIDGE>
```
6.2.1 XML Tags and Attributes

6.2.1.1 TNBRIDGE

This is the root tag and includes all the other tags. These are the following:

- **Connections**: contains a collection of Connection tags.
- **ScreenStyles**: contains a collection of ScreenStyle tags.
- **KeyboardMaps**: contains a collection of KeyboardMap tags.
- **HotSpots**: contains a collection of HotSpot tags.
- **TnbMacros**: contains a collection of Macro tags.

6.2.1.2 Connections

Contains a collection of **Connection** tags. Each **Connection** represents a particular connection and has the following tags:
- **Host**: specifies Host attributes.
- **Display**: specifies display options for the current connection.

**Attributes**

**Connections MRUName**
Indicates last connection name used.

**Syntax Example:**

```xml
<Connections MRUName="Clemson"/>
```

**Connection Name**
Indicates connection name.

**Syntax Example:**

```xml
<Connection Name="Clemson"/>
```

**Connection Type**
Indicates the type of the target system. Possible selection values are:

- IBM Mainframe (TN3270)
- IBM AS/400 (TN5250)

**Syntax Example**

```xml
<Connection Type="TN3270"/>
```

### 6.2.1.3 ScreenStyles
Contains a collection of **ScreenStyle** tags. Each **ScreenStyle** represents a particular screen style and has the following tags:

- **General**: contains general screen settings.
- **FieldMapping**: contains a collection of Fieldmaps.

**Attributes**

**ScreenStyles MRUName**
Indicates last screen style name used.

**Syntax Example:**

```xml
<ScreenStyles MRUName="Green"/>
```

**ScreenStyle Name**

Indicates the screen style name.

**Syntax Example:**

```xml
<ScreenStyle Name="Monocromatic"/>
```

### 6.2.1.4 KeyboardMaps

Contains a collection of **KeyboardMap** tags. Each **KeyboardMap** represents a particular keyboardmap and contains a collection of KeyMaps.

#### Attributes

**KeyboardMaps MRUName**

Indicates last keyboardmaps name used.

**Syntax Example:**

```xml
<KeyboardMaps MRUName="Default++"/>
```

**KeyboardMap Name**

Specifies the keyboard map's name.

**Syntax Example:**

```xml
<KeyboardMap Name="Default++"/>
```

**KeyboardMap Type**

Indicates the keyboard map type.

**Syntax Example:**

```xml
```
<KeyboardMap Type="3270"/>

### 6.2.1.5 HotSpots
Contains a collection of **HotSpotGroup** tags. Each **HotSpotGroup** contains a group of HotSpots.

**Attributes**

**HotSpots MRUName**
Indicates last hotspots name used.
**Syntax Example:**

```xml
<HotSpots MRUName="Clemson"/>
```

**HotSpotGroup Name**
Specifies the HotSpotGroup's name.
**Syntax Example:**

```xml
<HotspotGroup Name="Clemson"/>
```

**HotSpotGroup StartCol**
Specifies the HotSpotGroup's start column number coordinate.
**Syntax Example:**

```xml
<HotspotGroup StartCol="0"/>
```

**HotSpotGroup StartRow**
Specifies the HotSpotGroup's start row number coordinate.
**Syntax Example:**

```xml
<HotspotGroup StartRow="0"/>
```

**HotSpotGroup EndCol**
Specifies the HotSpotGroup's end column number coordinate.

**Syntax Example:**

```xml
<HotspotGroup EndCol="0"/>
```

**HotSpotGroup EndRow**

Specifies the HotSpotGroup's end row number coordinate.

**Syntax Example:**

```xml
<HotspotGroup EndRow="0"/>
```

**HotSpotGroup ViewAs**

Indicates the HotSpotGroup's shape. This shape can be: None, Plain, Button or Link.

**Syntax Example:**

```xml
<HotspotGroup ViewAs=""/>
```

---

### 6.2.1.6 TnbMacros

Contains a collection of **Macro** tags. Each **Macro** contains a set of **Actions** that indicates what the macro does.

#### Attributes

**Macro Version**

Indicates macro's version.

**Syntax Example:**

```xml
<macro version="1"/>
```

**Macro Name**

Specifies the Macro's name.

**Syntax Example:**

```xml
<macro Name="Save"/>
```
Macro DateTime

Specifies the date and time the macro was created.

**Syntax Example:**

```
<macro datetime="17/12/2004 15:19:23"/>
```

### 6.2.2 XML Example

Here you can see an example of a profile generated with XML Code:

```xml
<?xml version="1.0" ?>
<TNBRIDGE>
  <Connections MRUName="Clemson">
    <Connection Name="Clemson" Type="TN3270">
      <Host Address="clemson.clemson.edu" KeepAlive="False" Extended="False" />
      <Display Terminal="IBM-3278-2-E" />
    </Connection>
  </Connections>
  <ScreenStyles MRUName="Green">
    <ScreenStyle Name="Monocromatic">
      <General>
        <Font Auto="0" Size="9" ChSpacing="0" Name="Fixedsys" />
      </General>
      <FieldMapping>
        <FieldMap>
          <Input High="True" Unprotected="True" />
          <Output Color="White" />
        </FieldMap>
      </FieldMapping>
    </ScreenStyle>
  </ScreenStyles>
  <KeyboardMaps MRUName="Default++">
    <KeyboardMap Name="Default++" Type="3270">
      <KeyMap>
        <FunctionKey Name="Clear" />
      </KeyMap>
    </KeyboardMap>
  </KeyboardMaps>
</TNBRIDGE>
```
7 Purchasing TN Bridge Host Integration Pack

By purchasing any edition of TN Bridge Host Integration Pack you will gain access to technical support, free upgrades and updates and the activation of advanced features in your edition.

In this section you will find information regarding the different licensing options you have that will help you choose the type of order you need to place. Also this section explains how to place your order and finally activate your product so that you can enjoy all of the z/Scope benefits.

- Licensing Information
- How to Place an Order
- TN Bridge Registration
- Technical Support
7.1 Licensing Information

When it comes to purchasing TN Bridge Host Integration Pack, there are different editions available. Our wide range of possibilities assures you that you will make the best deal.

- TN Bridge Host Integration Pack for ActiveX
- TN Bridge Host Integration Pack for Delphi
- TN Bridge Host Integration Pack for DotNET*


Cybele Software offers perpetual licensing for all our products. The pricing will vary according to the TN Bridge Host Integration Pack package you choose (Developer/Team) and the amount of licenses.

We offer Technical Support by e-mail and/or phone, which also includes free updates and upgrades during the covered period and our full commitment to timely fix bugs and problems. The cost is 20% of the product’s value in advance, only mandatory for the first year. Thereafter, we encourage users to renew the annual maintenance contract in order to be eligible for technical support and product upgrades. The maintenance fee after the first year will still be 20% of the updated price of the purchased product.

Cybele Software offers volume pricing according to the amount of the purchase.

If you have any other question, contact us at sales@cybelesoft.com. Our sales representatives will get in touch with you to assist you with your licensing situation.

7.2 How to Place an Order

There are many ways to order your Tn Bridge licenses:

- Place an Online Order through our Web Site:

- Let us know about your licensing needs and we will send you an official quotation. Fill out the form in the following link:
  [http://www.cybelesoft.com/contact/](http://www.cybelesoft.com/contact/)

- Contact us at sales@cybelesoft.com. Our sales representatives will get in touch with you to assist you with the purchase.

- You can also call us anytime to any of these phone numbers and place the order
immediately:

**Toll Free: 1-866-462-9768**
Local line: 1-302-892-9625
Fax: 1-302-295-9995

- You can also contact us through Live Chat by pressing this icon in our website:

![Live Chat Icon](image)

and immediately have a conversation with a representative without even having to pick up the phone.

There are several payment options, and we also accept Purchase Orders.

When you buy TN Bridge, you will receive a Key to register the Trial version. For instructions on how to register TN Bridge when you purchase a license, see Registering TN Bridge Trial Version

### 7.3 How to register your License

If you downloaded TN Bridge Host Integration Pack's Trial Version from our web site or a distribution site and you have already purchased a license, you must follow these steps in order to register the product:

- **TN Bridge Host Integration Pack for ActiveX**
- **TN Bridge Host Integration Pack for Delphi**

#### 7.3.1 ActiveX Edition

Take these precautions before starting:

**Remember:** If you used a trial license, erase all the old files before entering the new one, to prevent license corruption.

**Important:** You will receive a Runtime license file only if you purchased additional Runtime Licenses.

- **Instructions for the Developer License:**
  The developer license file **TNBIPK3.LIC** must be saved (on the Development PC) into
the "windows" directory, and must
be also copied to the same directory where TNBridge was installed. Do not change the
file name.

- **Instructions for the Runtime License:**
The runtime license string included on the **TNBIPK3rt.txt** file must be added to the
program source code, as shown on
the examples included below:
The developer license file **TNBIPK3.LIC** must be saved into the "Windows" directory.

The runtime license string must be added to the program source code, as shown on
the examples included below:

- **C# example on how to embed the code (replace with you own license):**

```csharp
new Tnblib.TnbXLicense().Key = "Y2Ja9EstT8Tb+9hNhH28fDoVSWOLT7QV"+
   "igoceLrROVv3T+Kh-gyi4Gpd21BvQZh"+
   "5jDd6TbLyJNIKAFeP9b5UcekvaEoouDz"+
   "7iF9lQTIg-Q0HaeICm9OFznRM0xpSfnQ"+
   "m0f0ZAh7dmPL9eyt1bMIMA3n+xdCS3B"+
   "VfmGuzZgs6n9wvScvjd3E6UDNUuhFKZ9"+
   "lOuTc3H6M3Yzw1rb7gH5lmw2EvhHMSn"+
   "B5Tlz1x0oSP-6SYJ5ZoxiiL5er7jOjZw"+
   "36T7qF3n3am94YndwDFpKOCOFoUgex"+
   "ek8Tr19bfNfelnEcm7u5RE2aYAganL"+
   "qRuL3XY-ktdvnp1CUBwvJ";
```

- **VB/VB.NET example on how to embed the code (replace with you own
license):**

```vbnet
Dim License As Tnblib.TnbXLicense
License.Key = "Y2Ja9EstT8Tb+9hNhH28fDoVSWOLT7QV" + _
   "igoceLrROVv3T+Kh-gyi4Gpd21BvQZh" + _
   "5jDd6TbLyJNIKAFeP9b5UcekvaEoouDz" + _
   "7iF9lQTIg-Q0HaeICm9OFznRM0xpSfnQ" + _
   "m0f0ZAh7dmPL9eyt1bMIMA3n+xdCS3B" + _
   "VfmGuzZgs6n9wvScvjd3E6UDNUuhFKZ9" + _
   "lOuTc3H6M3Yzw1rb7gH5lmw2EvhHMSn" + _
   "B5Tlz1x0oSP-6SYJ5ZoxiiL5er7jOjZw" + _
   "36T7qF3n3am94YndwDFpKOCOFoUgex" + _
   "ek8Tr19bfNfelnEcm7u5RE2aYAganL" + _
   "qRuL3XY-ktdvnp1CUBwvJ";
```
7.3.2 Delphi Edition

The developer license file TNBIPK3.LIC must be saved into the "windows" directory.

**Important:** If you used a trial license, erase the old license files before entering the new one, to prevent license corruption.

**Instructions for the Developer License:**
The developer license file TNBIPK3.LIC must be saved into the "windows" directory, and must be also copied to the same directory where TNBridge was installed. Do not change the file name.

**Instructions for the Runtime License:** (only if you purchased additional Runtime Licenses)
The runtime license string must be added to the program source code, as shown on the example below:

```pascal
implementation
uses uTnbIPack;
...
...
initialization
LicenseKey := 'Y2Ja9EstTBt+9hNthH28fDoVSWOLT7QV'+'igoceLrR0UVyY3TvKh-gyi4Gpd21BvQZh'+'SjDd6TbjjNIKAFEpR9a5UcekvaEoouDz'+'7f9jQTIg-Q0HaeICm7OFznRM0xpSfnQ'+'m0f0ZhAH7dmPL9eyt1BIMaNn+xdCs3B'+'VfmGuzGzs69wv5cvjD3E6UDNuhFKZ9'+'IoUtc3H6M3YzwIrbd7gH5m2EvhHMSn'+'B51ZkkOoO-6SYJ5ZoxiL5er7jOjZw'+'36T7GfN3amk94YndwDfIPkOCTQFOsuex'+'ek8Tkl9bjFNehehEcM7u5RE2arYAganL'+'qRuL3XY-ktdvnp1CUBwvJ';
end;
```

* **Important:** If you used a trial license, erase all the old files before entering the new one, to prevent license corruption.

**Instructions for the Developer License:**
The developer license file TNBIPK3.LIC must be saved into the "windows" directory, and must be also copied to the same directory where TNBridge was installed. Do not change the file name.

**Instructions for the Runtime License:** (only if you purchased additional Runtime Licenses)
The runtime license string must be added to the program source code, as shown on the example below:

```pascal
implementation
uses uTnbIPack;
...
```
initialization
LicenseKey := 'Y2Ja9EstTBT+9hNtHH28fDoVSWOLT7QV'+'
'igoceLR0UvY3T+Kh-gyi4Gpd21BvQZh'+'
'5jDd6Tb]jNlKAFEpR9a5UcekvaEooouDz'+'
'7fF9lQTlg-Q0HaeICm9OFznRM0xpSnQf'+'
'm0f0ZhAH7dmPL9eytlbMIMaNn+xdC3sB'+'
'VfmGuzgs6n9wvScvjD3E6UDNUuhFKZ9'+'
'IOuTc3H6M3YZbwIr7gh5lww2EvhHMSn'+'
'B5TizxOoSP-6SjY3zoxiiL5er7jOjZw'+'
'36T7Gfn3am9YndwDfjpKOCOFouSgex'+'
'ek8Tl9bjfNfehnEcM7u5RE2arYAganL'+'
'qRuL3XY-ktdvnp1CUBwvJ'+'
end;

7.4 Obtaining Technical Support

Cybele's goal is to offer high quality products and services to increase the efficiency and ease-of-use of legacy systems. The whole Company focuses on this goal, and the results of our unique expertise are our reliable solutions. We believe passionately that modern, solid and feature-rich host access solutions can actually increase its users' productivity.

Technical support is a very important benefit to consider, especially when it comes to mission critical software solutions.

Using registered Cybele Software's applications not only allows you to receive free product upgrades and updates but also the certainty that you will have our team of experienced developers and technical support representatives working hard to assist you with any issue, thus making the product much more accessible in any situation.

We are here to help you out from Monday to Friday 9 a.m. to 5 p.m. eastern time on the phone numbers:

**Toll Free: 1-866-462-9768**
Local line: 1-302-892-9625
Fax: 1-302-295-9995

If you make your call outside this hour range, you can leave a message and we will get back to you.

You can send us an email to [support@cybelesoft.com](mailto:support@cybelesoft.com) and we will write you back timely. You can also contact us through Live Chat by pressing this icon in our website:

![Live Help - Online](click for live support)

and immediately have a conversation with a representative without even having to pick up the phone.