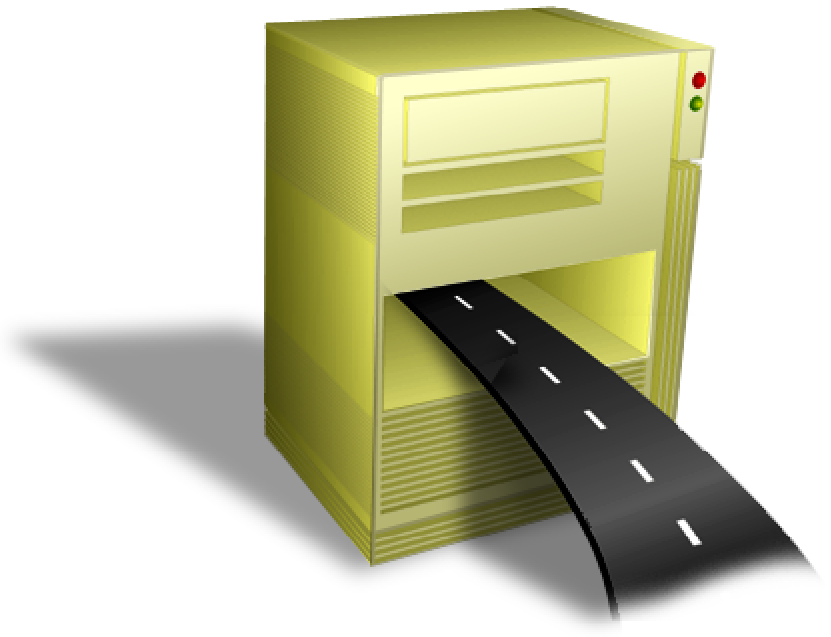


TN Bridge Host Integration Pack

Developer's guide

Table of Contents

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.

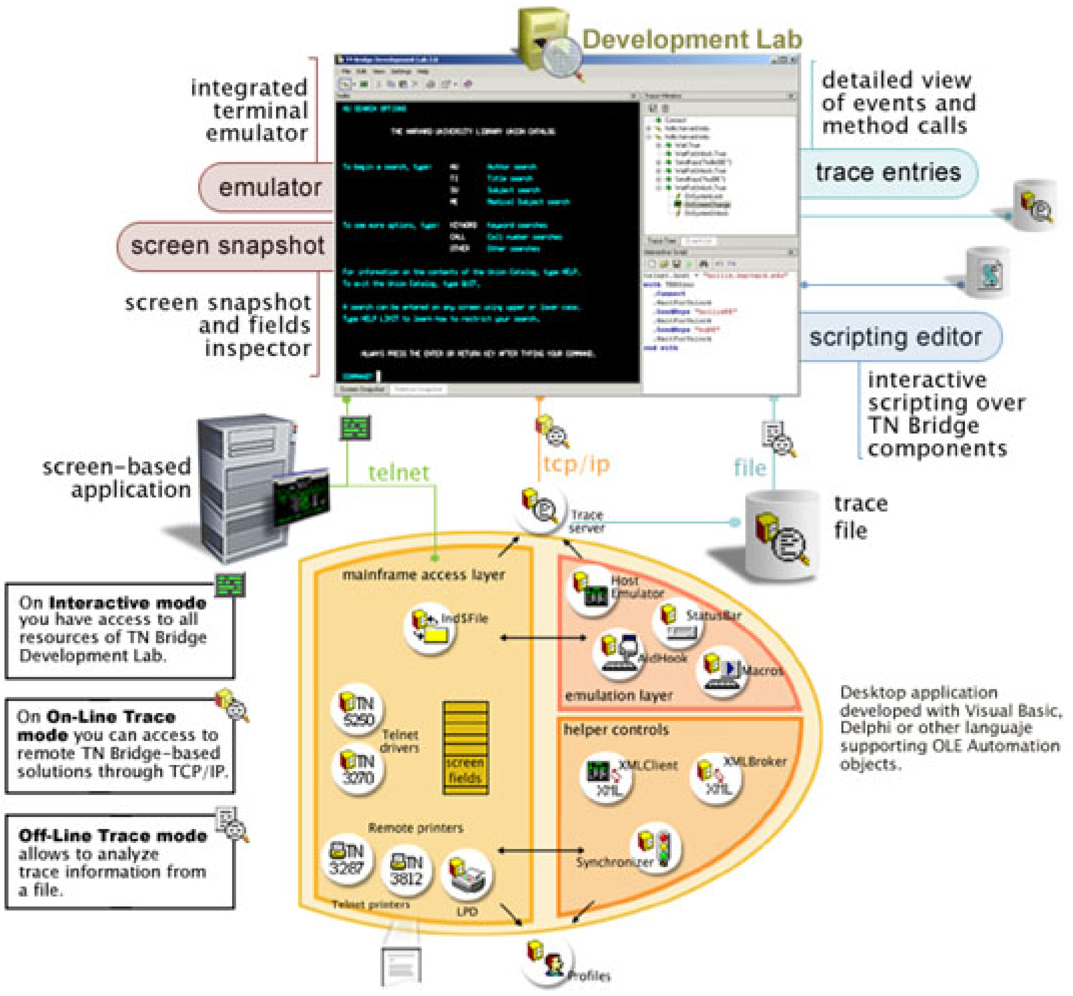
Includes a productivity environment that helps during the design and development-cycle of host-integrated applications. This tool drastically reduces the training curve and testing time.

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 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.


Development Lab
integrated terminal emulator
emulator
screen snapshot
screen snapshot and fields inspector
detailed view of events and method calls
trace entries
scripting editor
interactive scripting over TN Bridge components
screen-based application
telnet
tcp/ip
file
Trace server
trace file
On **Interactive mode** you have access to all resources of TN Bridge Development Lab.
On **On-Line Trace mode** you can access to remote TN Bridge-based solutions through TCP/IP.
Off-Line Trace mode allows to analyze trace information from a file.
mainframe access layer
IndSFile
TN 5250
TN 3270
Telnet drivers
screen fields
Remote printers
TN 3287
TN 3812
Telnet printers
LPD
Host Emulator
StatusBar
AidHook
Macros
emulation layer
helper controls
XMLClient
XML
XMLBroker
XML
Synchronizer
Profiles
Desktop application developed with Visual Basic, Delphi or other languaje supporting OLE Automation objects.

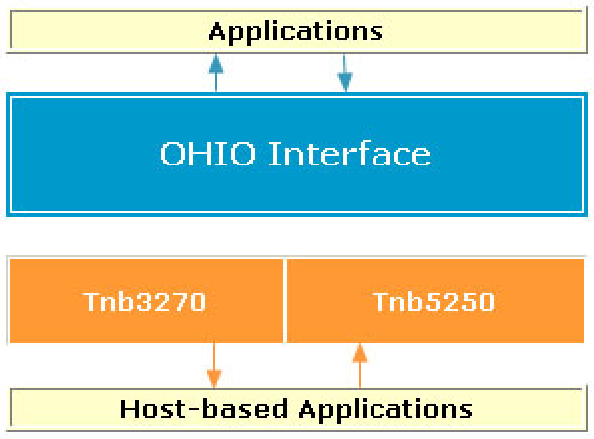
1.3 What's New in 3.5?

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:

http://www.cybelesoft.com/download/register.aspx?product=HIPA

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 purchsed 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.

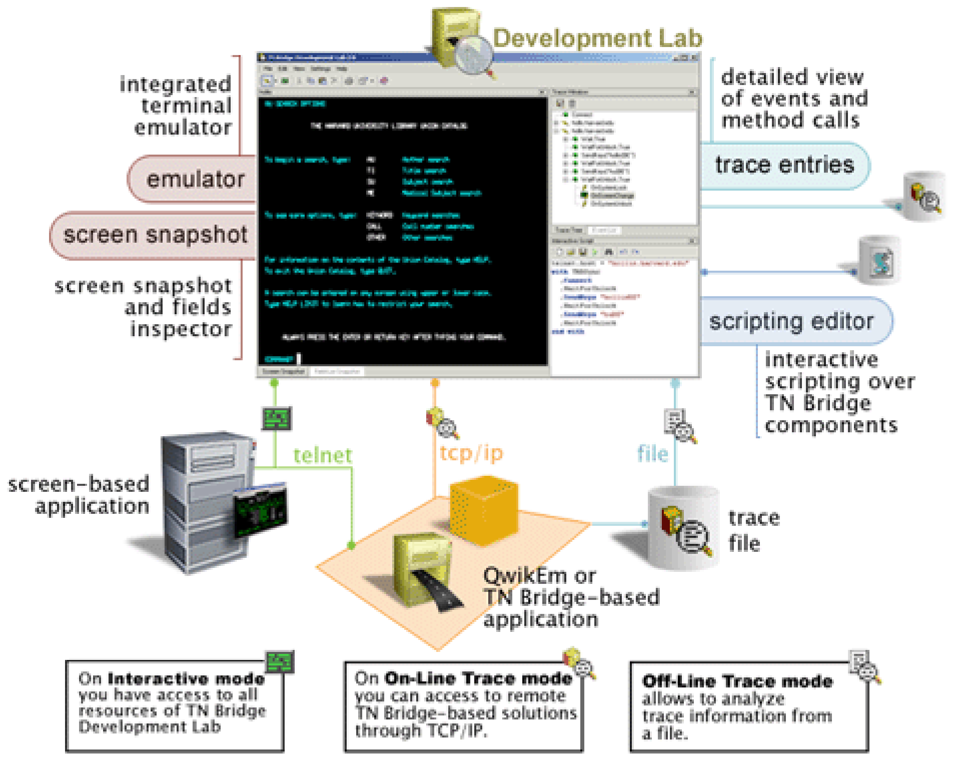
TN BRIDGE Development Lab will help you develop your applications in these scenarios:

- 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**.

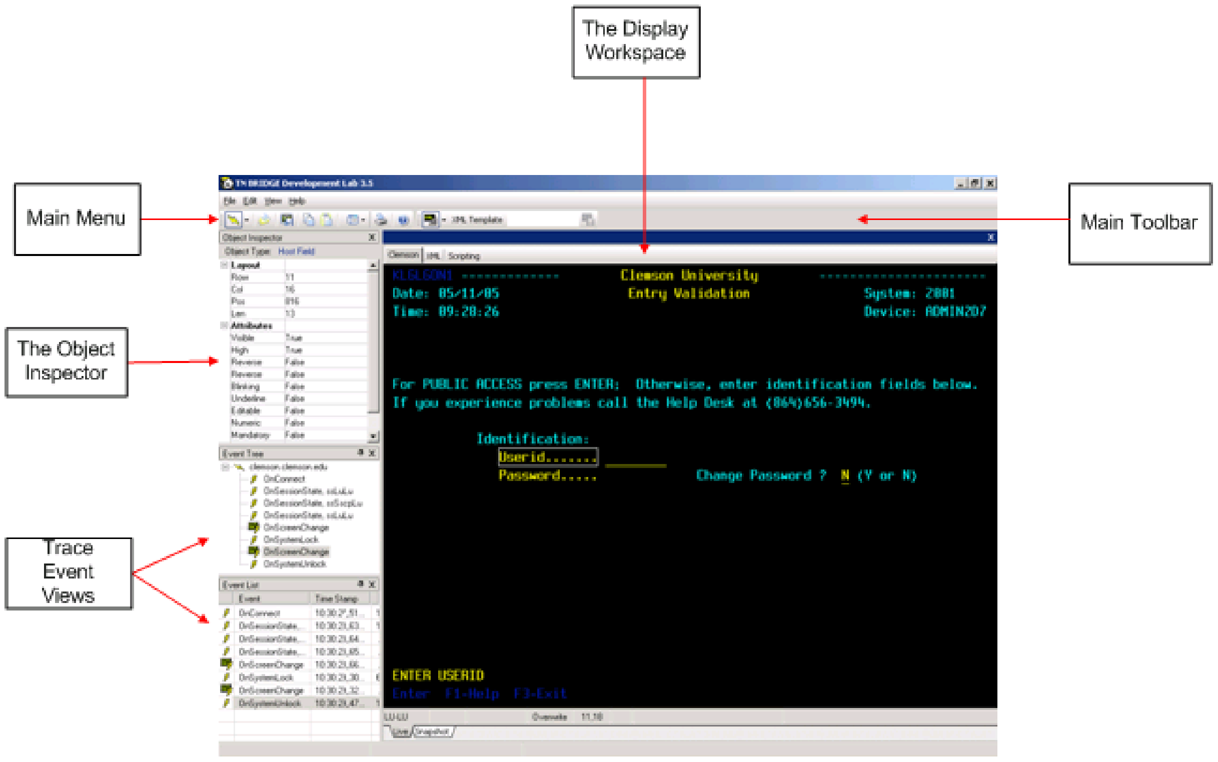
You can also **save screens as XML files** and connect to them to work with mainframe screens being off-line.


Development Lab
integrated terminal emulator
emulator
screen snapshot
screen snapshot and fields inspector
detailed view of events and method calls
trace entries
scripting editor
interactive scripting over TN Bridge components
telnet
tcp/ip
file
screen-based application
trace file
QwikEm or TN Bridge-based application
On Interactive mode you have access to all resources of TN Bridge Development Lab
On On-Line Trace mode you can access to remote TN Bridge-based solutions through TCP/IP.
Off-Line Trace mode allows to analyze trace information from a file.

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.

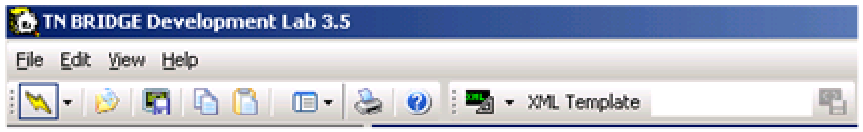


```
KLGLGON1 -------------         Clemson University          -----------------------
Date: 05/11/05                  Entry Validation                    System: 2001
Time: 09:28:26                                                      Device: ADMIN207


For PUBLIC ACCESS press ENTER;  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

           Identification:
             Userid.......  _______
             Password.....              Change Password ?  N (Y or N)


ENTER USERID
Enter  F1-Help  F3-Exit
```

2.1.1 Main Menu and Toolbar

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

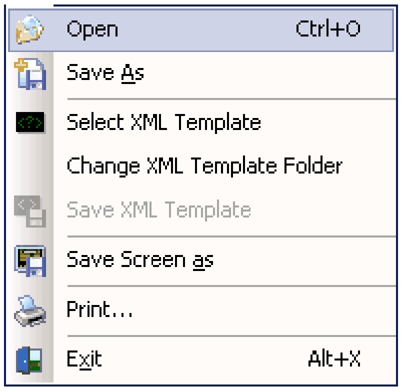
- **Main Menu** offers four submenus for accessing to actions and settings.
- **Main Toolbar** The toolbar buttons offer quick access to the most commonly used functions.



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

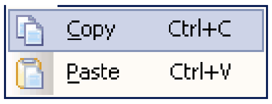
Change XML Template Folder
Use this option to display the standard printer setup dialog.

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.

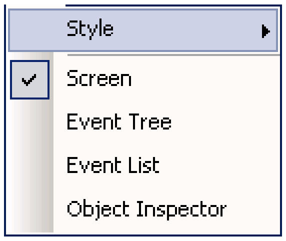
Edit



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



Style
Allows you to choose a different Application Skin.

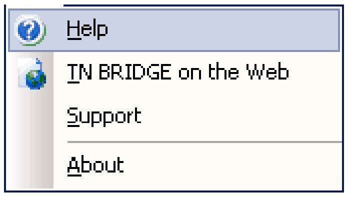
Screen
Shows/Hides the display window

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



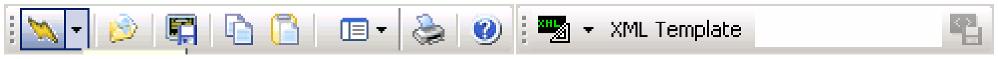
Help
Use this option to get access to this help.

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

Help
Use this button to get access to this help.

XML Template

Enable/Disable Template Mode

Save XML Template file

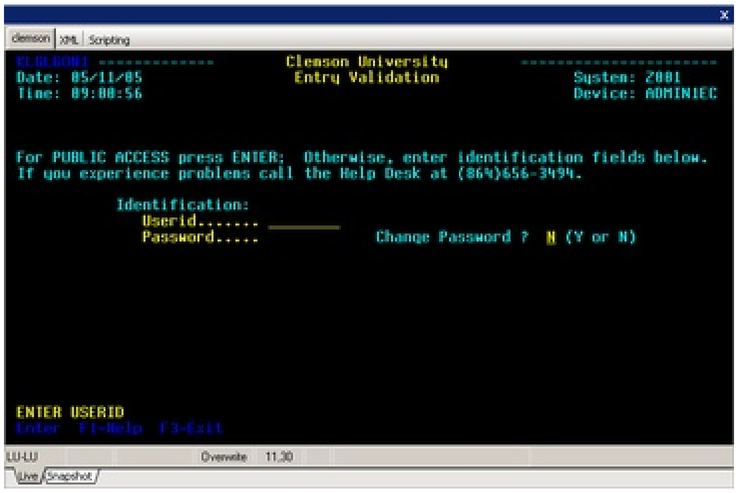
2.1.2 The Display workspace

The display workspace allows access to the following windows:

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

2.1.2.1 Interactive emulation

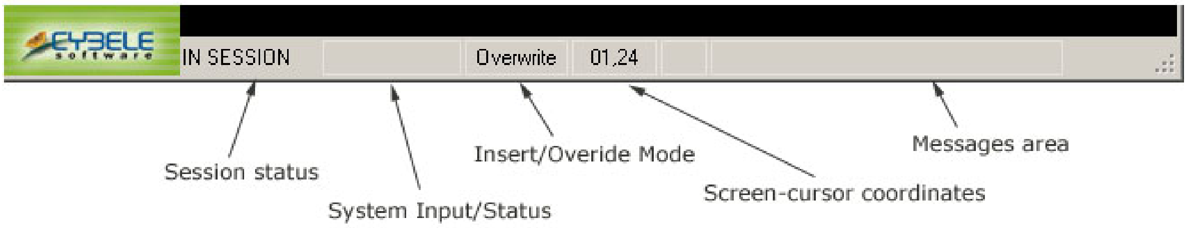
The interactive emulation window allows you to get connected to your mainframe and take the mainframe connection as source of tracing windows, XML transforming and interactive scripting.



```
KLBLGON1 --------------        Clemson University          -----------------------
Date: 05/11/05                  Entry Validation                  System: Z001
Time: 09:00:56                                                    Device: ADMINIEC


For PUBLIC ACCESS press ENTER;  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

             Identification:
                Userid.......
                Password.....              Change Password ?  N (Y or N)







ENTER USERID
Enter   F1-Help   F3-Exit
```

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 bellow:

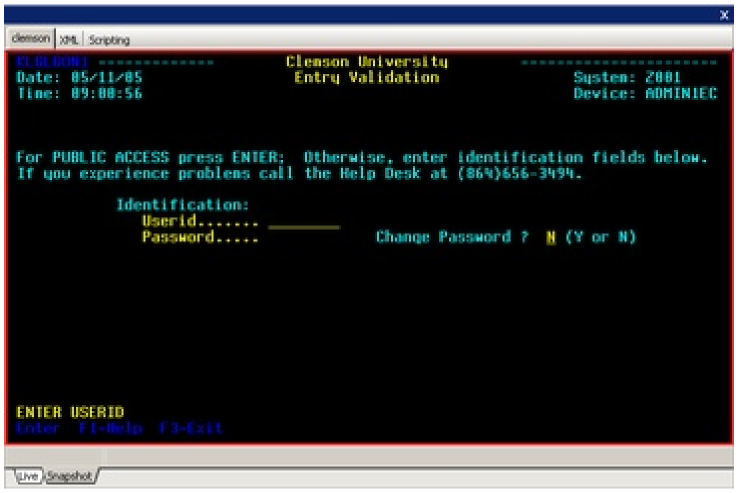
- Connection status.
- System/Input status.
- Insert/Overwrite mode.
- Screen-cursor coordinates.
- Messages area.



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.



```
KLBLGON1 --------------          Clemson University          ----------------------
Date: 05/11/05                    Entry Validation                    System: Z001
Time: 09:00:56                                                        Device: ADMINIEC


For PUBLIC ACCESS press ENTER:  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

             Identification:
                Userid.......
                Password.....            Change Password ?  N (Y or N)










ENTER USERID
Enter  F1-Help  F3-Exit
```

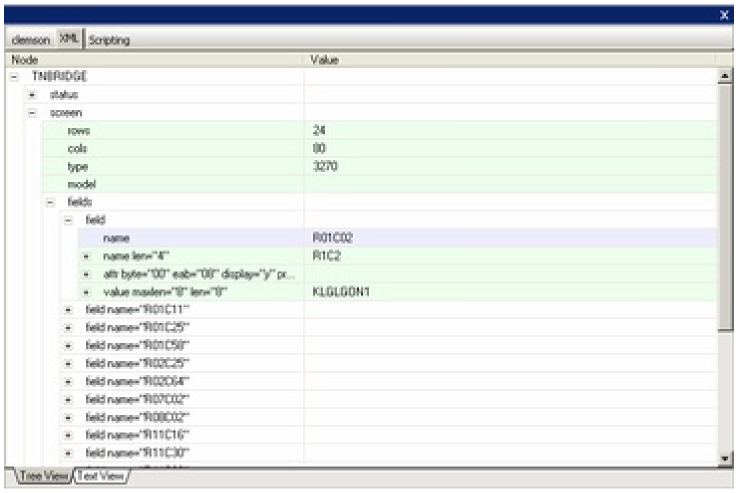
See also **Trace Event views**

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.

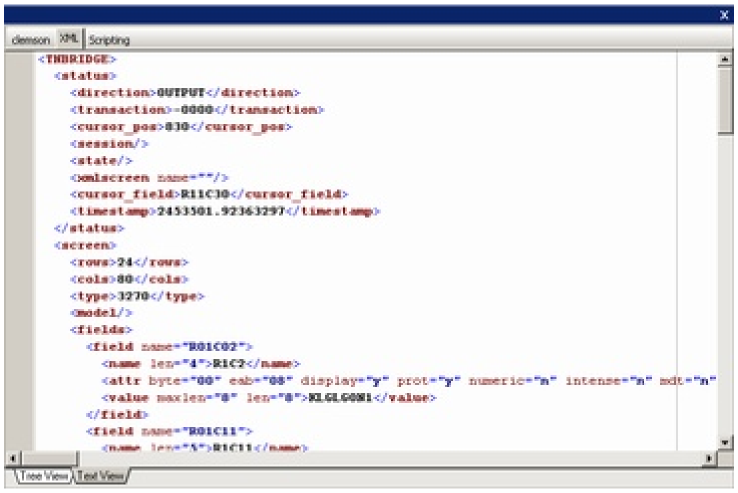
Two views are available:

- XML Tree view:



Node	Value
TNBRIDGE	
status	
screen	
rows	24
cols	80
type	3270
model	
fields	
field	
name	R01C02
name len="4"	R1C2
attr byte="00" eab="00" display="y" pr...	
value maxlen="8" len="8"	KLGLGON1
field name="R01C11"	
field name="R01C25"	
field name="R01C50"	
field name="R02C25"	
field name="R02C64"	
field name="R07C02"	
field name="R08C02"	
field name="R11C16"	
field name="R11C30"	

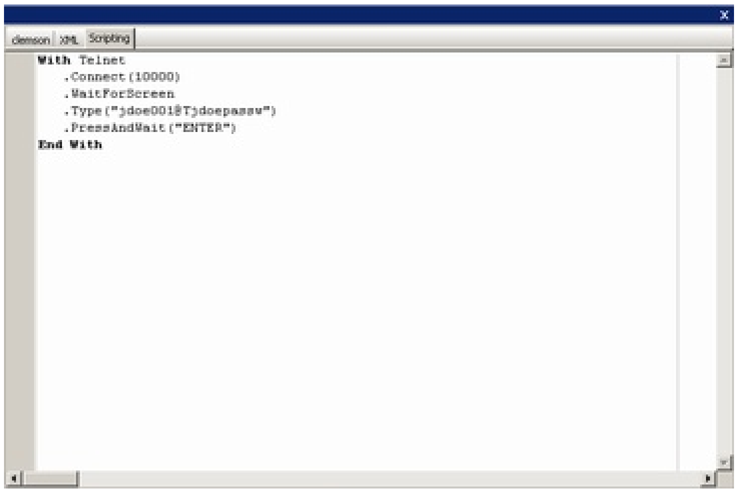
- XML Text view



```
<TNBRIDGE>
  <status>
    <direction>OUTPUT</direction>
    <transaction>-0000</transaction>
    <cursor_pos>830</cursor_pos>
    <session/>
    <state/>
    <xmlscreen name=""/>
    <cursor_field>R11C30</cursor_field>
    <timestamp>2453501.92363297</timestamp>
  </status>
  <screen>
    <rows>24</rows>
    <cols>80</cols>
    <type>3270</type>
    <model/>
    <fields>
      <field name="R01C02">
        <name len="4">R1C2</name>
        <attr byte="00" 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>
```

2.1.2.4 Scripting window

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



```
With Telnet
    .Connect(10000)
    .WaitForScreen
    .Type("jdoe001@Tjdoepassw")
    .PressAndWait("ENTER")
End With
```

2.1.3 The Trace Event views

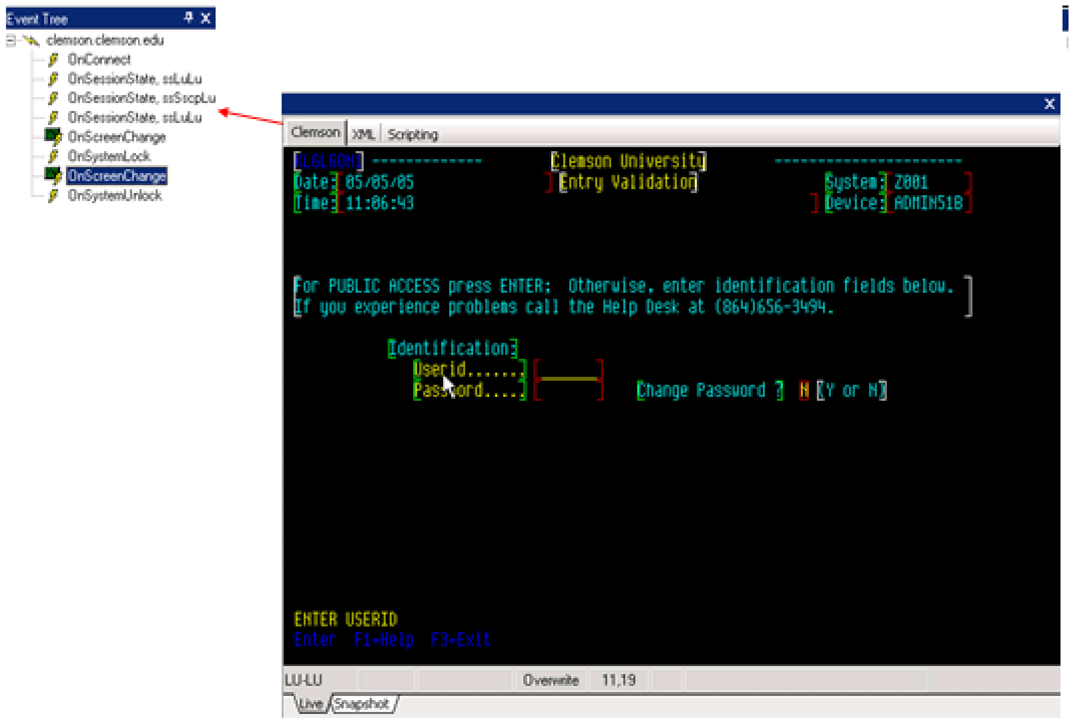
The Trace window shows the events generated by TN BRIDGE in relation to the communication with the mainframe. This information can be gathered from the interactive emulation, from an external application using the TN BRIDGE components or from file.

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.



```
clemson.clemson.edu
    OnConnect
    OnSessionState, ssLuLu
    OnSessionState, ssSscpLu
    OnSessionState, ssLuLu
    OnScreenChange
    OnSystemLock
    OnScreenChange
    OnSystemUnlock
```

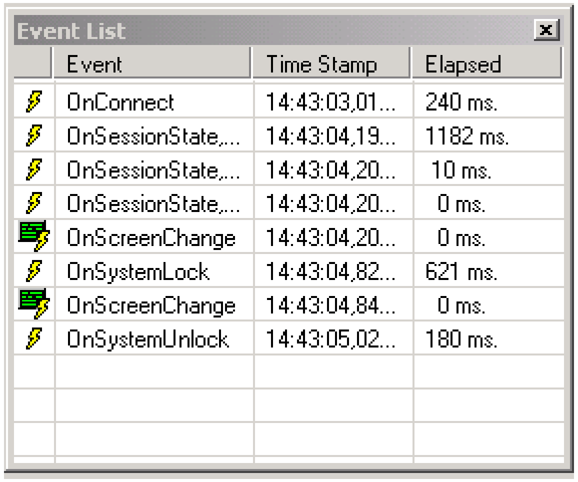
```
KLGLGON1 -------------          Clemson University        -----------------------
Date: 05/05/05                  Entry Validation                  System: Z081
Time: 11:06:43                                                    Device: ADMIN51B


For PUBLIC ACCESS press ENTER:  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

           Identification:
              Userid.......
              Password.....              Change Password ?  N (Y or N)







ENTER USERID
Enter  F1=Help  F3=Exit
```

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 procesed during the session. It also displays additional information such us the exact time of the event reception and the elapsed time it took to be triggered.

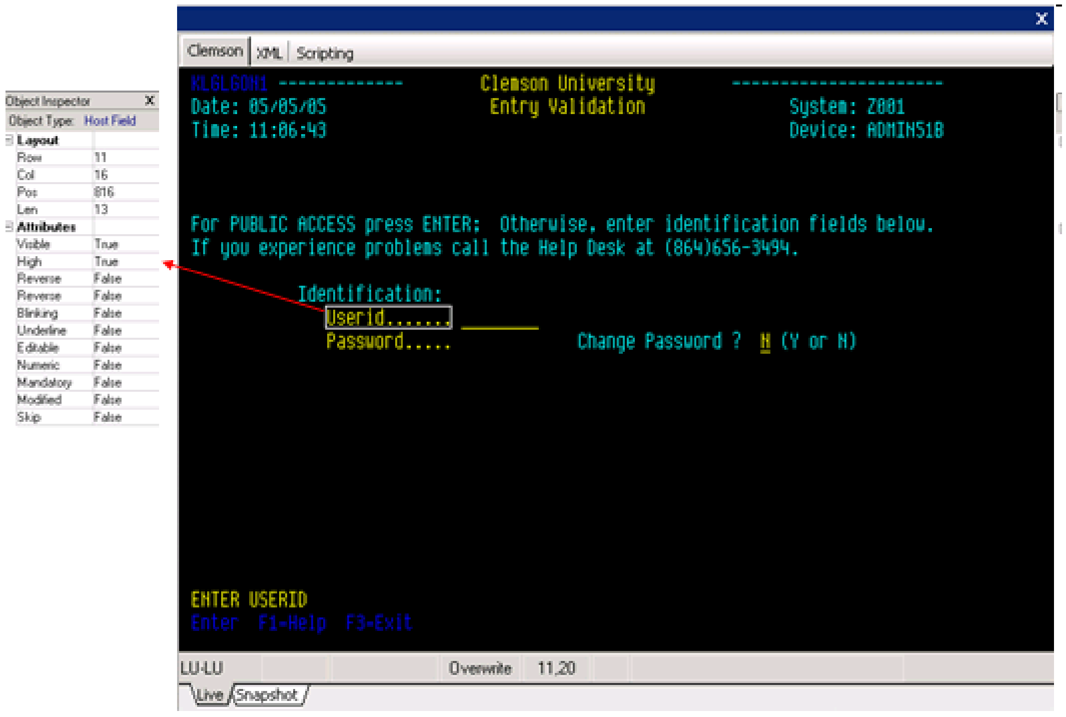


	Event	Time Stamp	Elapsed
	OnConnect	14:43:03,01...	240 ms.
	OnSessionState,...	14:43:04,19...	1182 ms.
	OnSessionState,...	14:43:04,20...	10 ms.
	OnSessionState,...	14:43:04,20...	0 ms.
	OnScreenChange	14:43:04,20...	0 ms.
	OnSystemLock	14:43:04,82...	621 ms.
	OnScreenChange	14:43:04,84...	0 ms.
	OnSystemUnlock	14:43:05,02...	180 ms.

2.1.4 The Object Inspector

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

- To inspect the attributes of an specific field, double-click on it.



Object Type: Host Field

Layout	
Row	11
Col	16
Pos	816
Len	13
Attributes	
Visible	True
High	True
Reverse	False
Reverse	False
Blinking	False
Underline	False
Editable	False
Numeric	False
Mandatory	False
Modified	False
Skip	False

```
KLGLGON1 -------------        Clemson University        ----------------------
Date: 05/05/05                 Entry Validation                  System: Z001
Time: 11:06:43                                                   Device: ADMIN51B


For PUBLIC ACCESS press ENTER:  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

           Identification:
              Userid.......  ________
              Password.....             Change Password ?  N (Y or N)









ENTER USERID
Enter  F1=Help  F3=Exit
```

See **XMLTemplates** for information about XML fields.

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 Develoment Lab modes

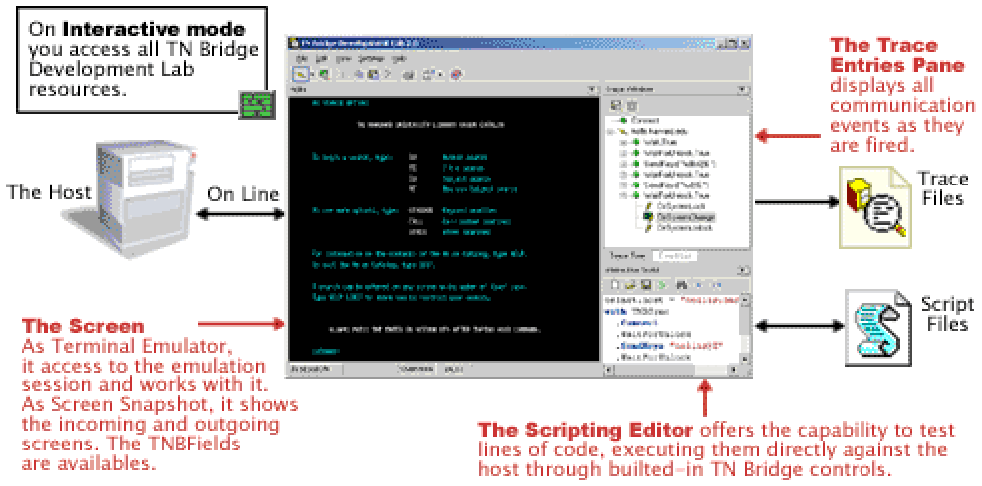
Main use of TN BRIDGE Development Lab is tracing TN BRIDGE-based applications. It provides three modes of operation: interactive, on-line trace and off-line trace.

- **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

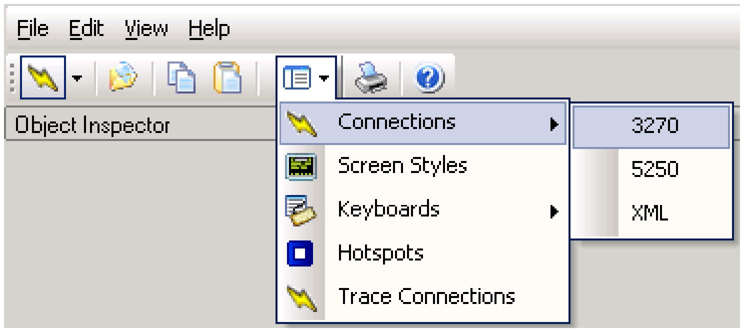
This mode provides a built-in terminal emulator which can be used to get connected to a Tn3270 or Tn5250 host. Addiionally, it allows to use XML files to simulate mainframe screens.

- **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.

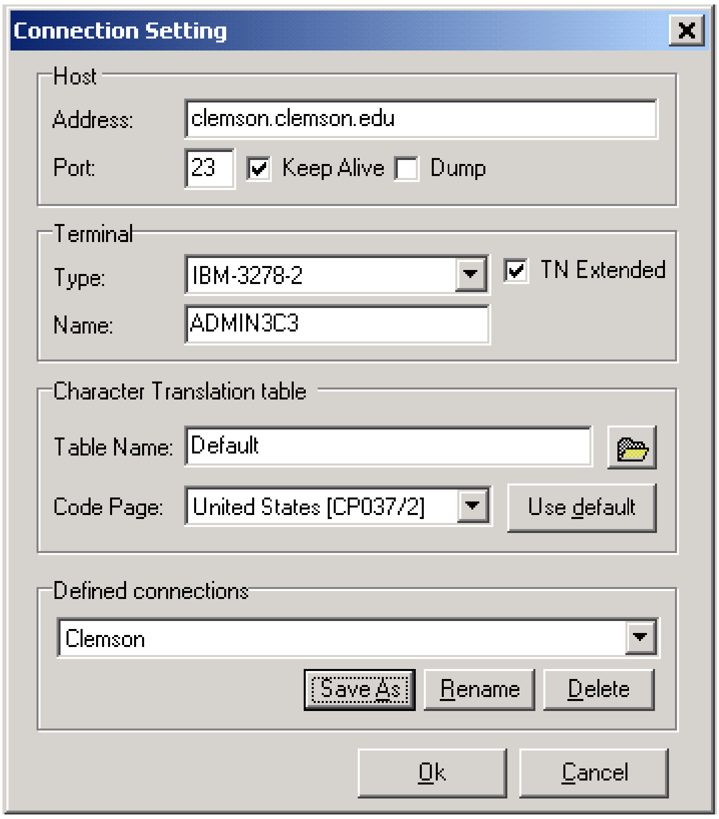

On **Interactive mode** you access all TN Bridge Development Lab resources.
The Host
On Line
The Trace Entries Pane displays all communication events as they are fired.
Trace Files
Script Files
The Screen As Terminal Emulator, it access to the emulation session and works with it. As Screen Snapshot, it shows the incoming and outgoing screens. The TNBFields are availables.
The Scripting Editor offers the capability to test lines of code, executing them directly against the host through builted-in TN Bridge controls.

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:



Address: clemson.clemson.edu
Port: 23 [✓ Keep Alive] Dump
Type: IBM-3278-2 [✓ TN Extended]
Name: ADMIN3C3
Table Name: Default
Code Page: United States [CP037/2]
Defined connections: Clemson

After filling in the appropiate 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.

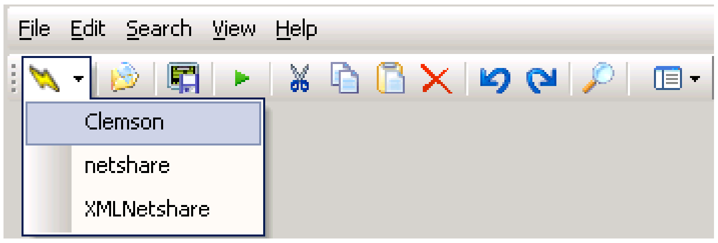
Code Page
Allows the selection of an internal character conversion table.

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

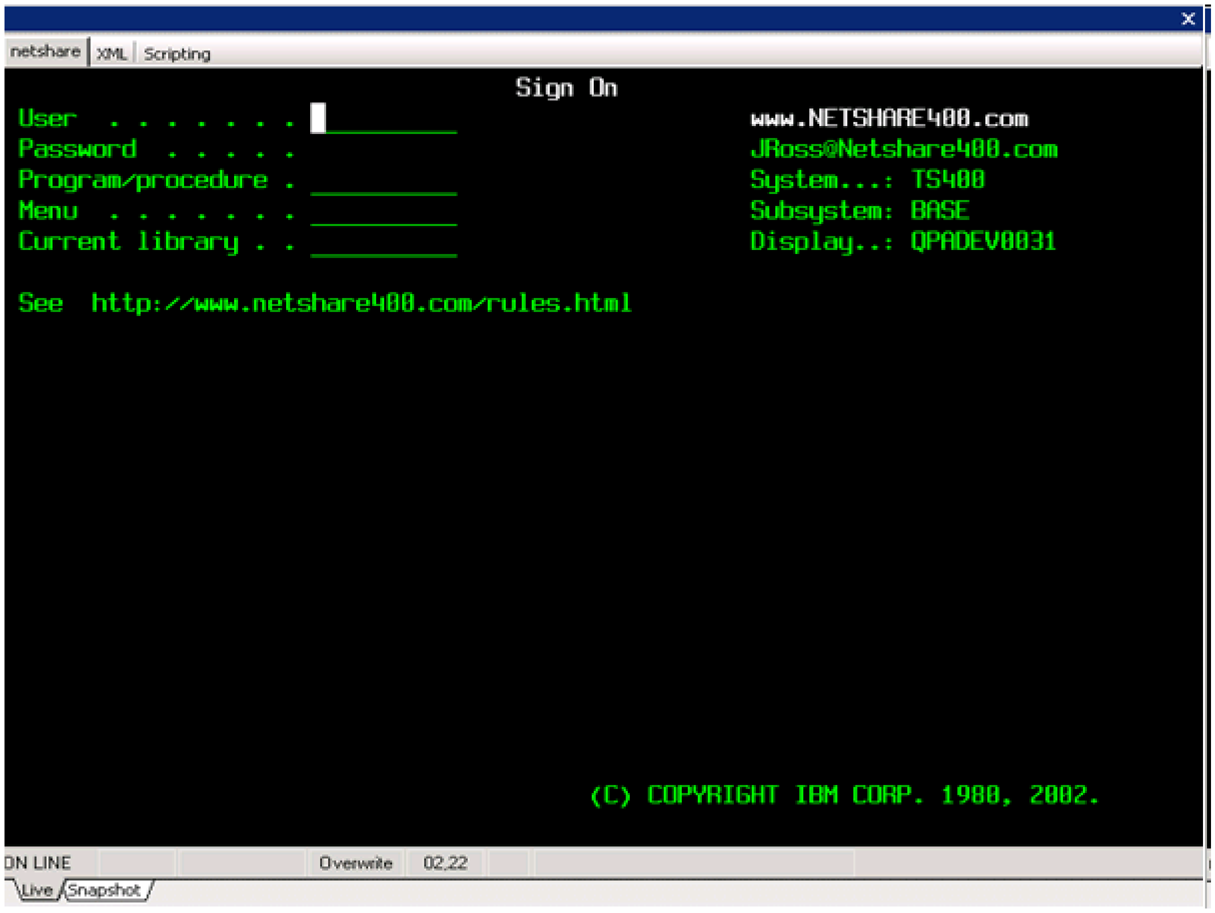
XML connections allows you work with mainframe screens in an off-line fashion. To work in this mode, you need to create the XML files corresponding to the mainframe application you want to work with.

- **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

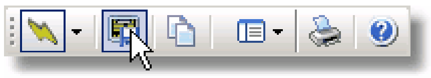
To save a screen in XML format you must do the following:

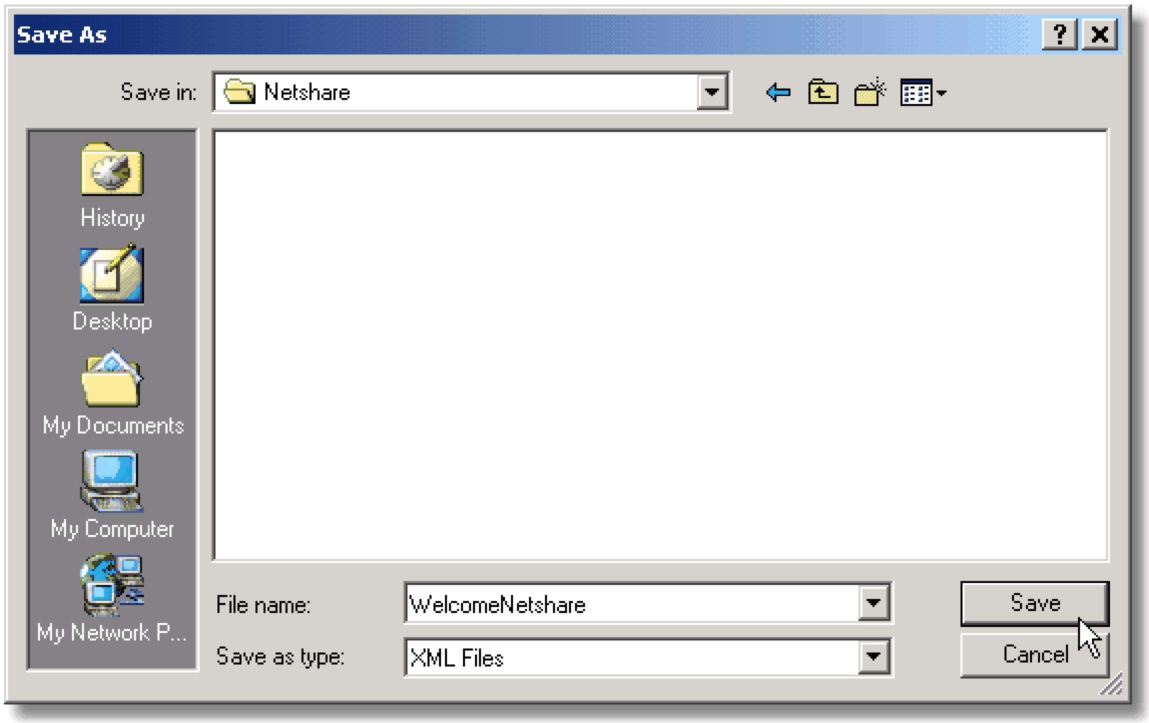
- Go to the mainframe screen you want to save



```
                                  Sign On
User  . . . . . . . .                              www.NETSHARE400.com
Password  . . . . . .                              JRoss@Netshare400.com
Program/procedure . ___________                    System...: TS400
Menu  . . . . . . . . ___________                  Subsystem: BASE
Current library . . ___________                    Display..: QPADEV0031

See  http://www.netshare400.com/rules.html

                                        (C) COPYRIGHT IBM CORP. 1980, 2002.
```

- Click on Save Screen button





File name: WelcomeNetshare

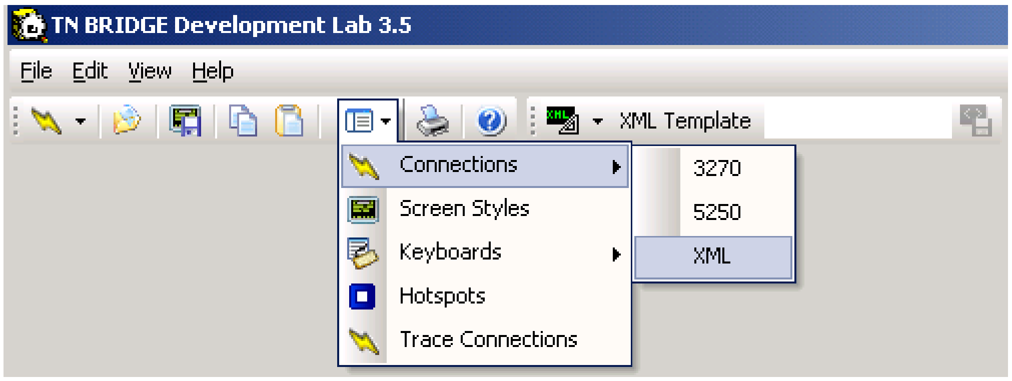
Save as type: XML Files

- Repeat the operation for all the screens you want to see off-line.

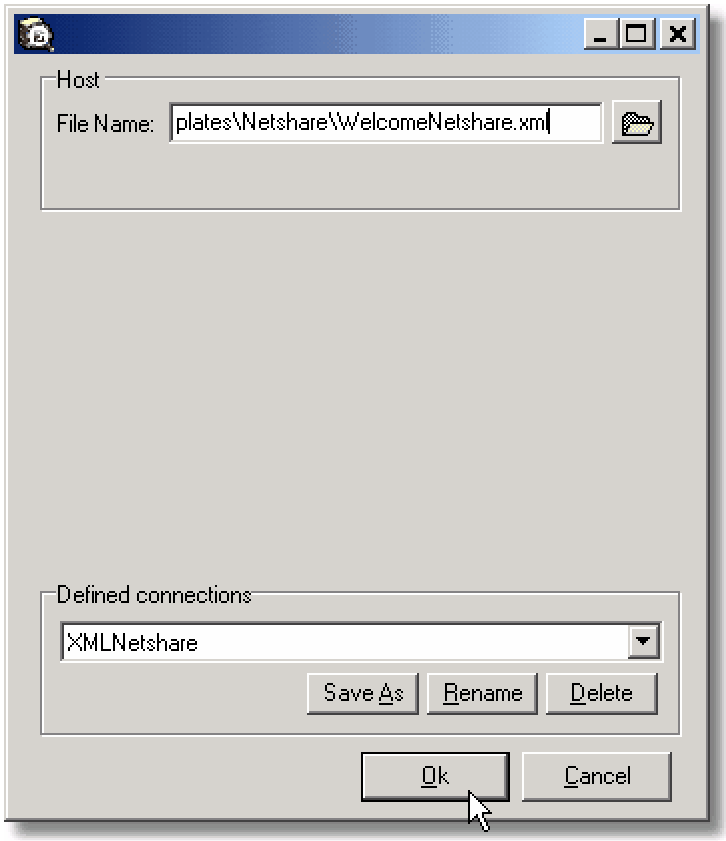
2.2.1.1.3.2 Connecting to a set of XML screen files

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

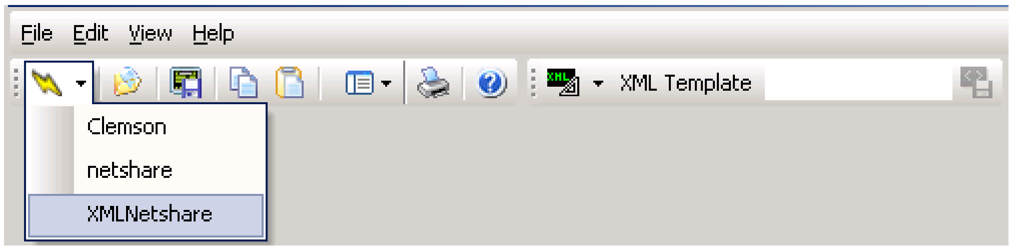

Host

File Name: plates\Netshare\WelcomeNetshare.xml

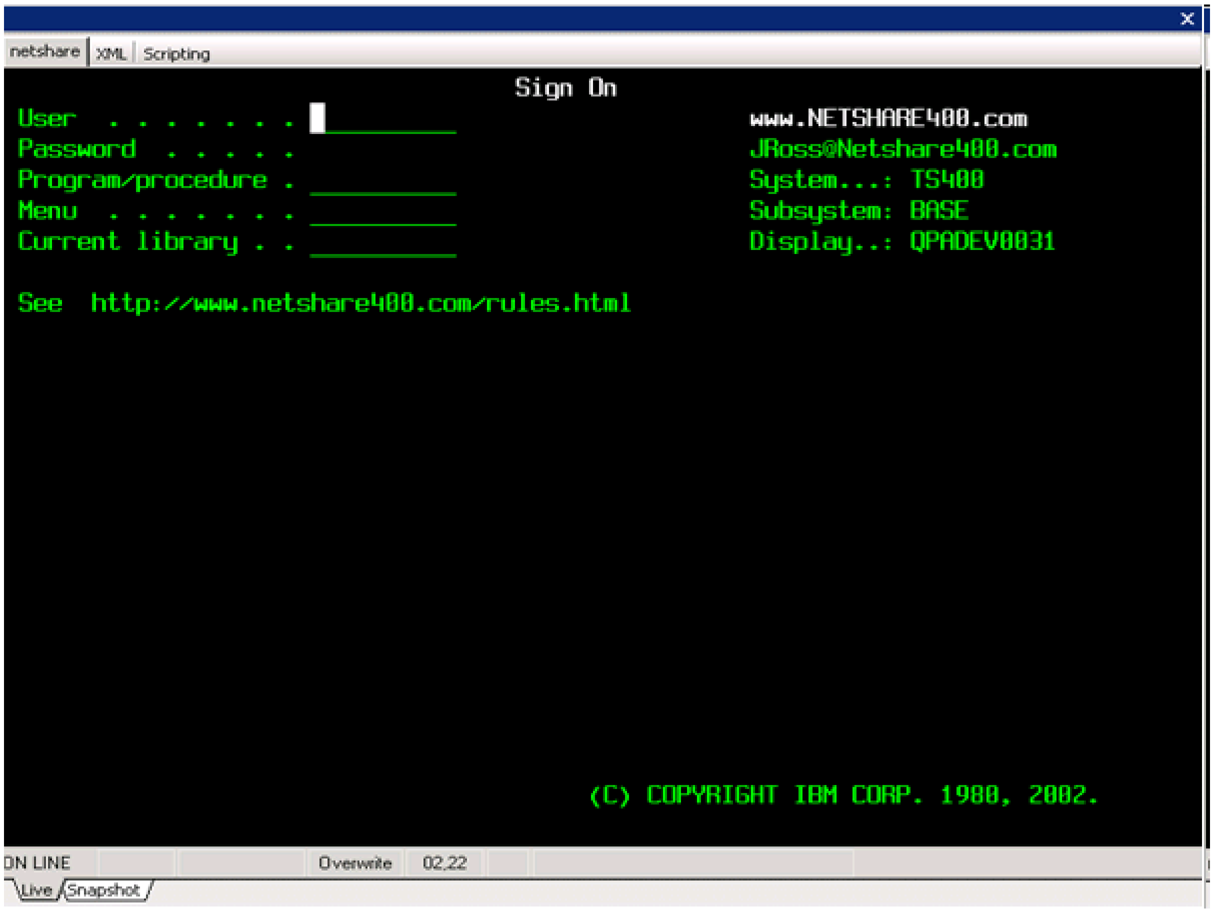
Defined connections

XMLNetshare

3. To connect, click on your defined XML connection.



4. If the screen is displayed, you are having a successful XML offline session.



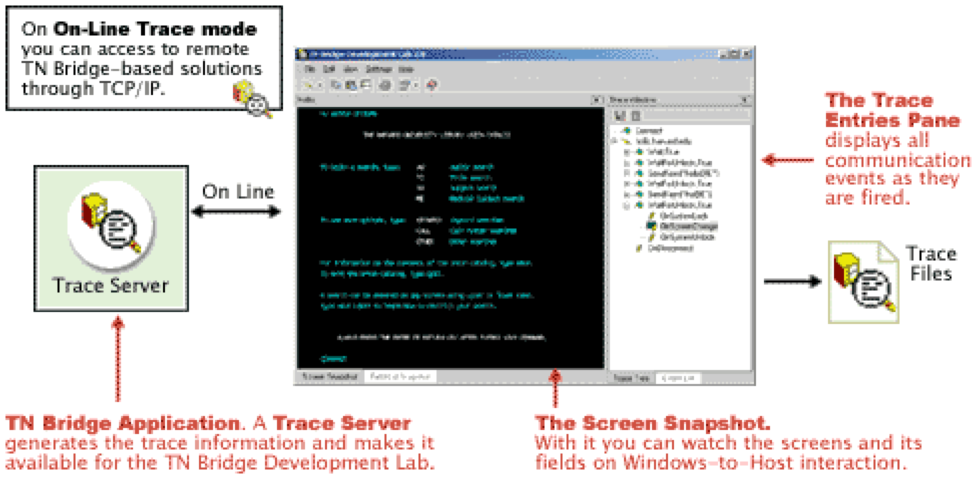
```
                                    Sign On
User . . . . . . . . ▌__________                www.NETSHARE400.com
Password . . . . . .                            JRoss@Netshare400.com
Program/procedure . __________                  System...: TS400
Menu . . . . . . . . __________                 Subsystem: BASE
Current library . . __________                  Display..: QPADEV0031

See  http://www.netshare400.com/rules.html




                                        (C) COPYRIGHT IBM CORP. 1980, 2002.
```

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.

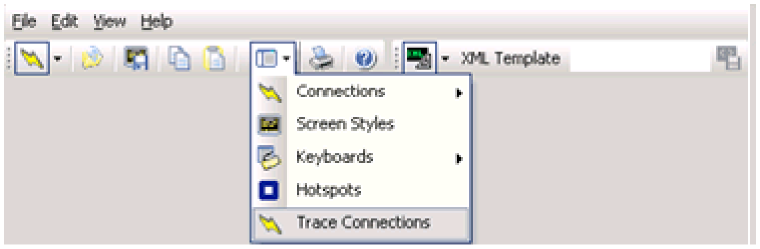
- **Connecting to a TN BRIDGE Application** Shows how to trace a TN BRIDGE application running in your computer or even on a computer within your LAN.


On On-Line Trace mode you can access to remote TN Bridge-based solutions through TCP/IP.
Trace Server
On Line
The Trace Entries Pane displays all communication events as they are fired.
Trace Files
TN Bridge Application. A Trace Server generates the trace information and makes it available for the TN Bridge Development Lab.
The Screen Snapshot. With it you can watch the screens and its fields on Windows-to-Host interaction.

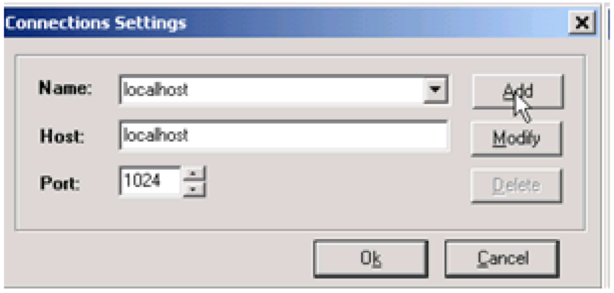
2.2.1.2.1 Connecting to a TN BRIDGE Application

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.



Name: localhost
Host: localhost
Port: 1024

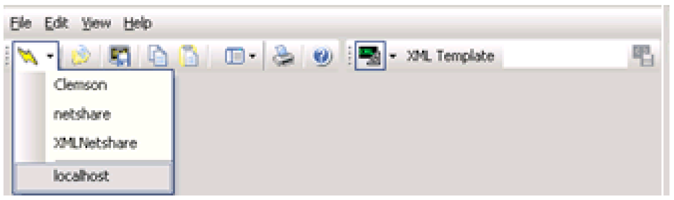
Parameters

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.

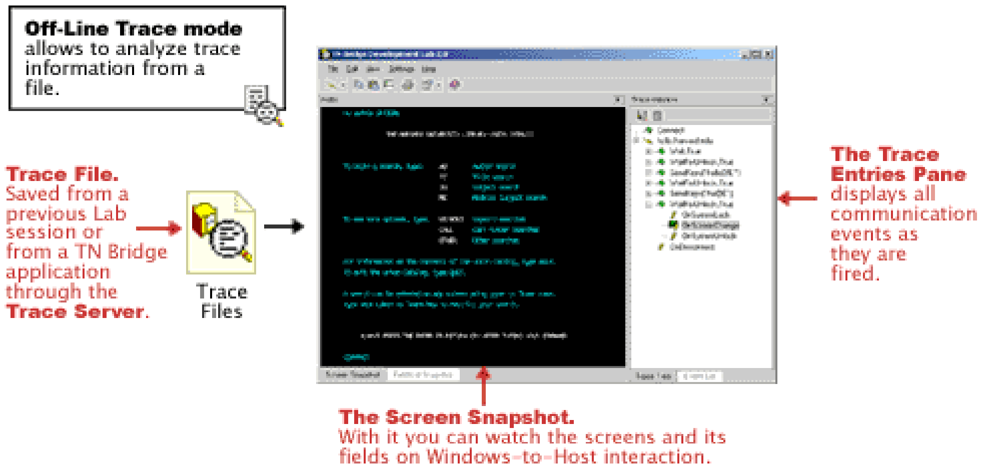
- You are now ready to connect to a remote TN BRIDGE Application



2.2.1.3 Off-Line tracing

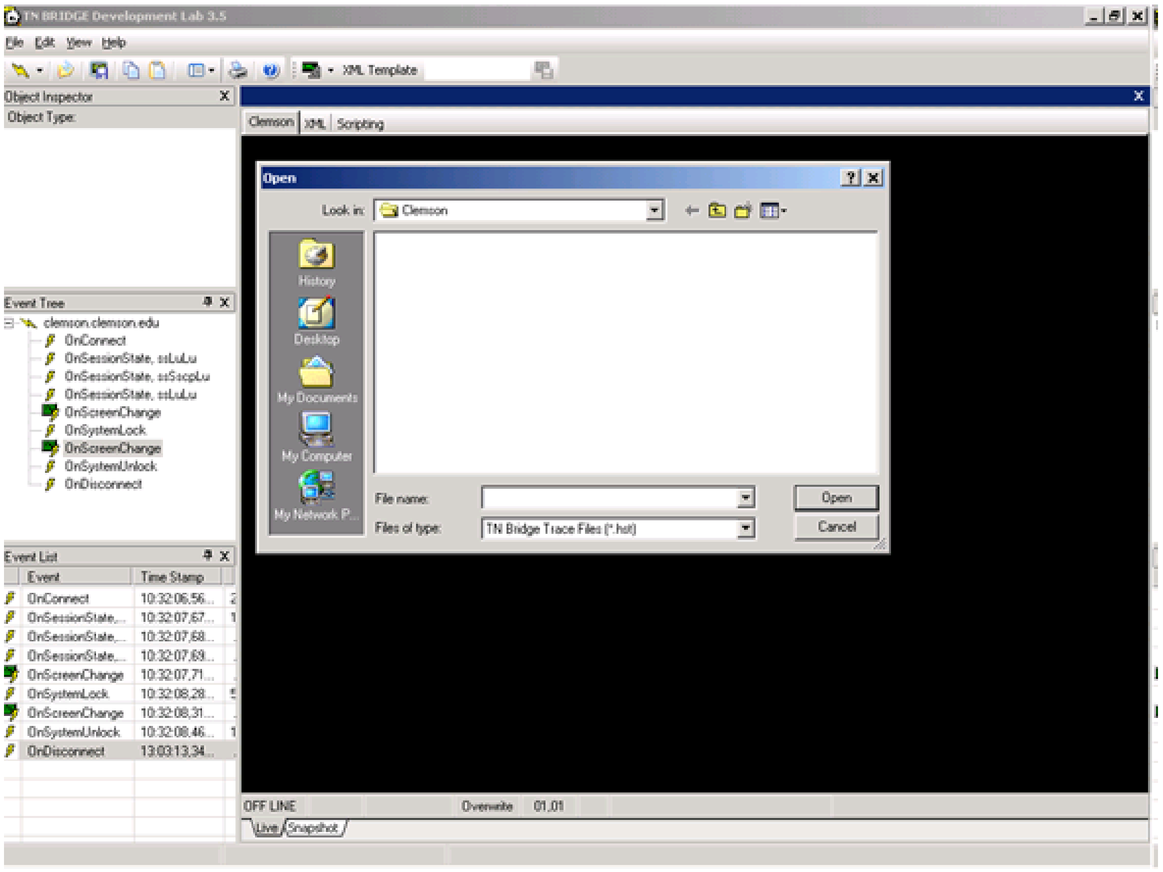
TN BRIDGE Development Lab allows you open trace files.

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


Off-Line Trace mode allows 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.**
Trace Files
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.



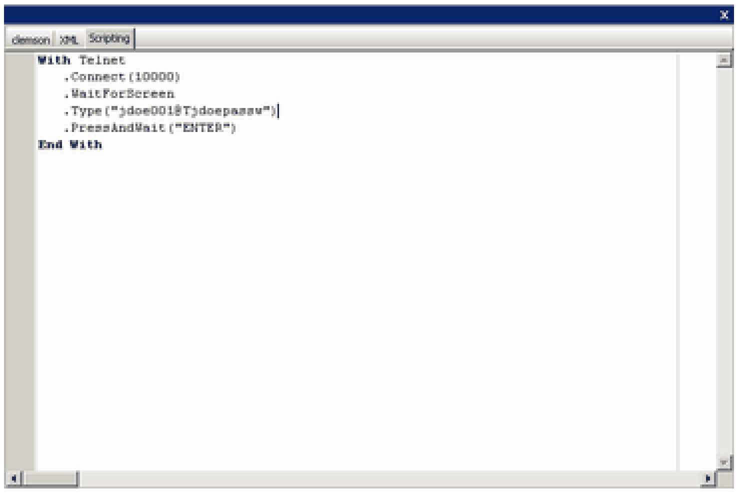
Event	Time Stamp
OnConnect	10:32:06,56...
OnSessionState,...	10:32:07,67...
OnSessionState,...	10:32:07,68...
OnSessionState,...	10:32:07,69...
OnScreenChange	10:32:07,71...
OnSystemLock	10:32:08,28...
OnScreenChange	10:32:08,31...
OnSystemUnlock	10:32:08,46...
OnDisconnect	13:03:13,34...

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.

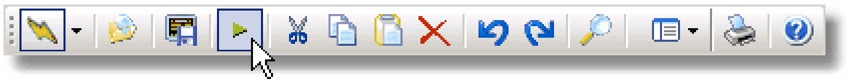
Code can be written in Microsoft VB Script language, accessing methods and properties of Tn3270/Tn5250 components. A global object "Telnet" is available, associated to the Tn3270 or Tn5250 currently in use in the interactive emulation.

As an example, you can write the following code access a mainframe application:

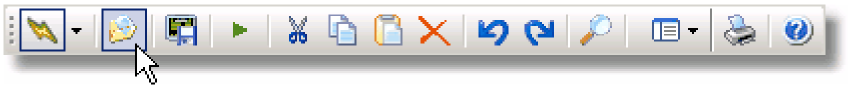


```
With Telnet
    .Connect(10000)
    .WaitForScreen
    .Type("jdoe001@Tjdoepassw")
    .PressAndWait("ENTER")
End With
```

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 top provide customized XML tags an 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 propery 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:

```
  <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">
                            <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>
```

2.2.3.2 XML Templates

What is an XML Template?

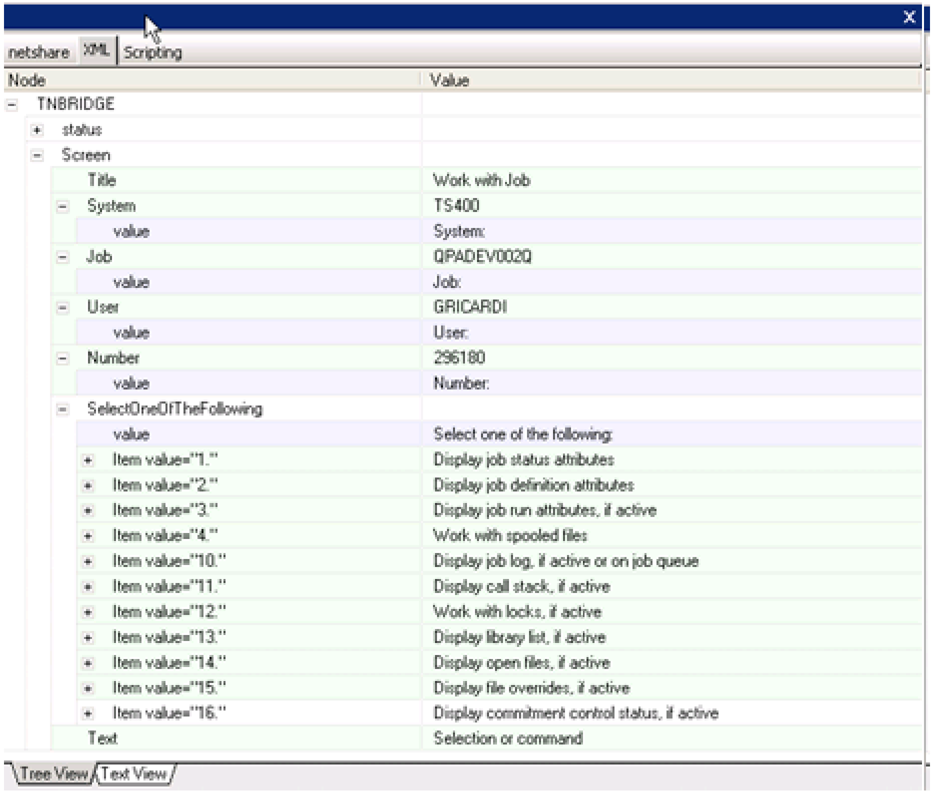
By default, Host Integration Pack provides a **generic XML** representation. Because this generic XML contains complete screen information it is useful to recreate a screens on the comsumer side. However, when it comes to real business information, it doesn't provides a real advantage regarding to accessing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

Using Develpement 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



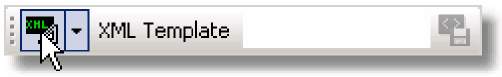
Node	Value
TNBRIDGE	
status	
Screen	
Title	Work with Job
System	TS400
value	System:
Job	QPADEV002Q
value	Job:
User	GRICARDI
value	User:
Number	296180
value	Number:
SelectOneOfTheFollowing	
value	Select one of the following:
Item value="1."	Display job status attributes
Item value="2."	Display job definition attributes
Item value="3."	Display job run attributes, if active
Item value="4."	Work with spooled files
Item value="10."	Display job log, if active or on job queue
Item value="11."	Display call stack, if active
Item value="12."	Work with locks, if active
Item value="13."	Display library list, if active
Item value="14."	Display open files, if active
Item value="15."	Display file overrides, if active
Item value="16."	Display commitment control status, if active
Text	Selection or command

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.

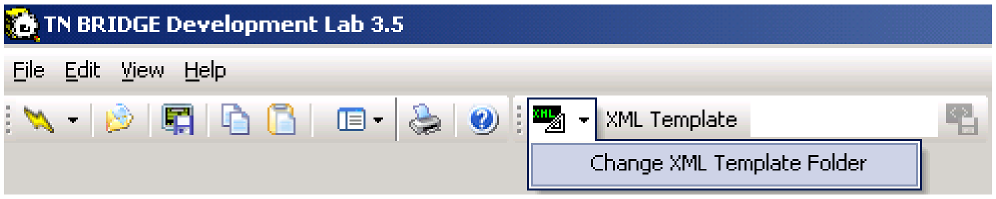
- Click on the icon as shown below.



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 tediuos 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 rutime.
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 doesnt 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.

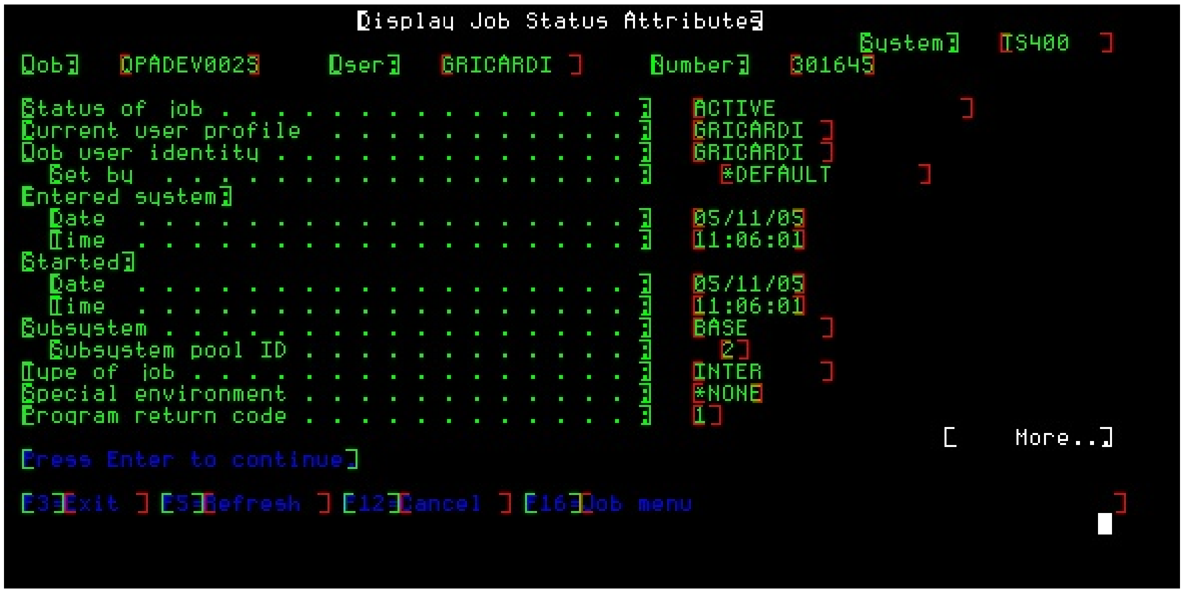
For example, in:

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.

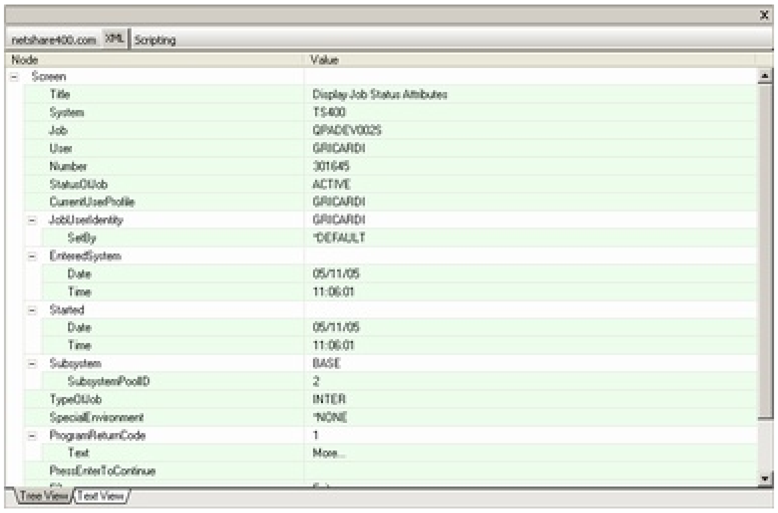
The resulting template fields are shown:
- Label fields enclosed in green brackets
- Text fields enclosed in red brackets
- Label & Text fields enclosed in white brackets



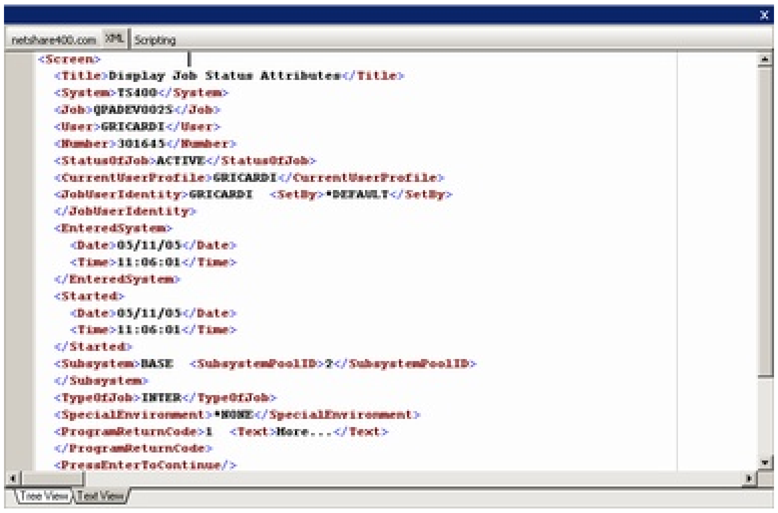
```
                         Display Job Status Attributes
                                                        System:   TS400   ]
Job:   QPADEV002S     User:   GRICARDI ]     Number:   301645

Status of job . . . . . . . . . . . . . . . :   ACTIVE            ]
Current user profile  . . . . . . . . . . . :   GRICARDI ]
Job user identity . . . . . . . . . . . . . :   GRICARDI ]
  Set by  . . . . . . . . . . . . . . . . . :     *DEFAULT      ]
Entered system:
  Date  . . . . . . . . . . . . . . . . . . :   05/11/05
  Time  . . . . . . . . . . . . . . . . . . :   11:06:01
Started:
  Date  . . . . . . . . . . . . . . . . . . :   05/11/05
  Time  . . . . . . . . . . . . . . . . . . :   11:06:01
Subsystem . . . . . . . . . . . . . . . . . :   BASE     ]
  Subsystem pool ID . . . . . . . . . . . . :     2]
Type of job . . . . . . . . . . . . . . . . :   INTER    ]
Special environment . . . . . . . . . . . . :   *NONE
Program return code . . . . . . . . . . . . :   1]
                                                                [     More..]
Press Enter to continue]

F3=Exit ]  F5=Refresh ]  F12=Cancel ]  F16=Job menu                           ]
```

*Note: You can now use the **Object Inspector** to customize the field you need.*

Switching to XML view we get:



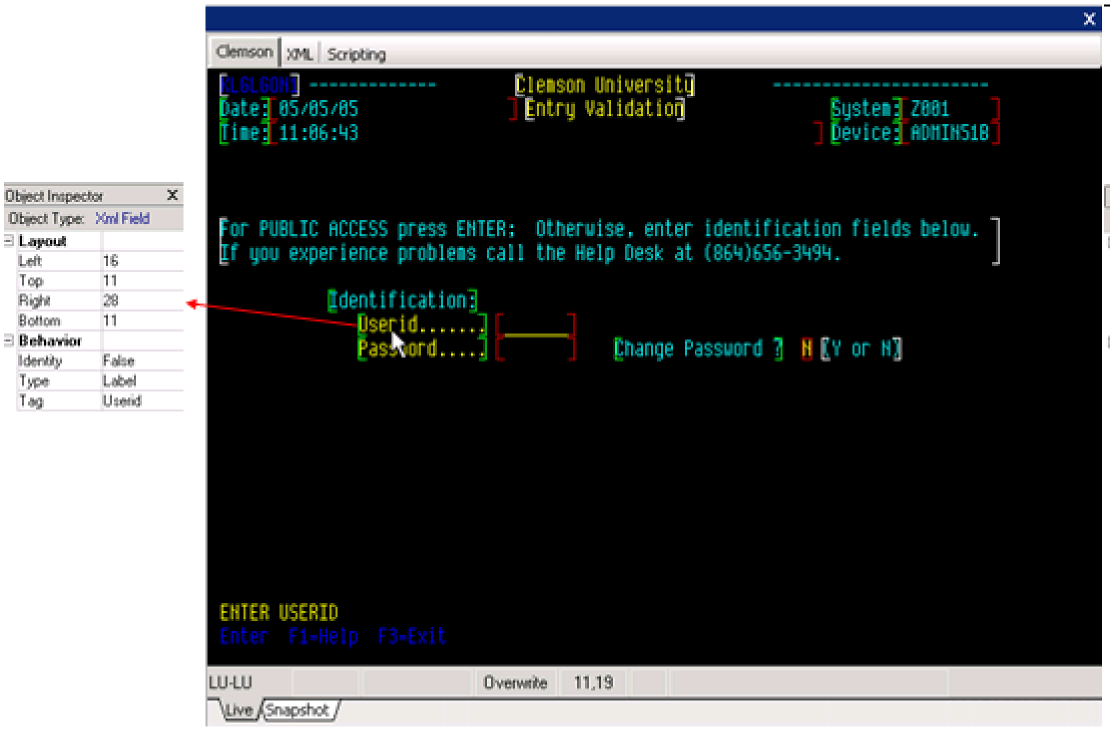
Node	Value
Screen	
Title	Display Job Status Attributes
System	TS400
Job	QPADEV002S
User	GRICARDI
Number	301645
StatusOfJob	ACTIVE
CurrentUserProfile	GRICARDI
JobUserIdentity	GRICARDI
SetBy	*DEFAULT
EnteredSystem	
Date	05/11/05
Time	11:06:01
Started	
Date	05/11/05
Time	11:06:01
Subsystem	BASE
SubsystemPoolID	2
TypeOfJob	INTER
SpecialEnvironment	*NONE
ProgramReturnCode	1
Text	More...
PressEnterToContinue	



```
<Screen>
  <Title>Display Job Status Attributes</Title>
  <System>TS400</System>
  <Job>QPADEV0025</Job>
  <User>GRICARDI</User>
  <Number>301645</Number>
  <StatusOfJob>ACTIVE</StatusOfJob>
  <CurrentUserProfile>GRICARDI</CurrentUserProfile>
  <JobUserIdentity>GRICARDI  <SetBy>*DEFAULT</SetBy>
  </JobUserIdentity>
  <EnteredSystem>
    <Date>05/11/05</Date>
    <Time>11:06:01</Time>
  </EnteredSystem>
  <Started>
    <Date>05/11/05</Date>
    <Time>11:06:01</Time>
  </Started>
  <Subsystem>BASE  <SubsystemPoolID>2</SubsystemPoolID>
  </Subsystem>
  <TypeOfJob>INTER</TypeOfJob>
  <SpecialEnvironment>*NONE</SpecialEnvironment>
  <ProgramReturnCode>1  <Text>More...</Text>
  </ProgramReturnCode>
  <PressEnterToContinue/>
```

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.



Object Inspector

Object Type: Xml Field

Layout	
Left	16
Top	11
Right	28
Bottom	11
Behavior	
Identity	False
Type	Label
Tag	Userid

```
XLGLGON1 -------------          Clemson University          ----------------------
Date: 05/05/05                  Entry Validation                  System: Z001
Time: 11:06:43                                                    Device: ADMIN51B



For PUBLIC ACCESS press ENTER:  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

           Identification:
               Userid.......
               Password.....              Change Password ?  N  (Y or N)









ENTER USERID
Enter  F1=Help  F3=Exit
```

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

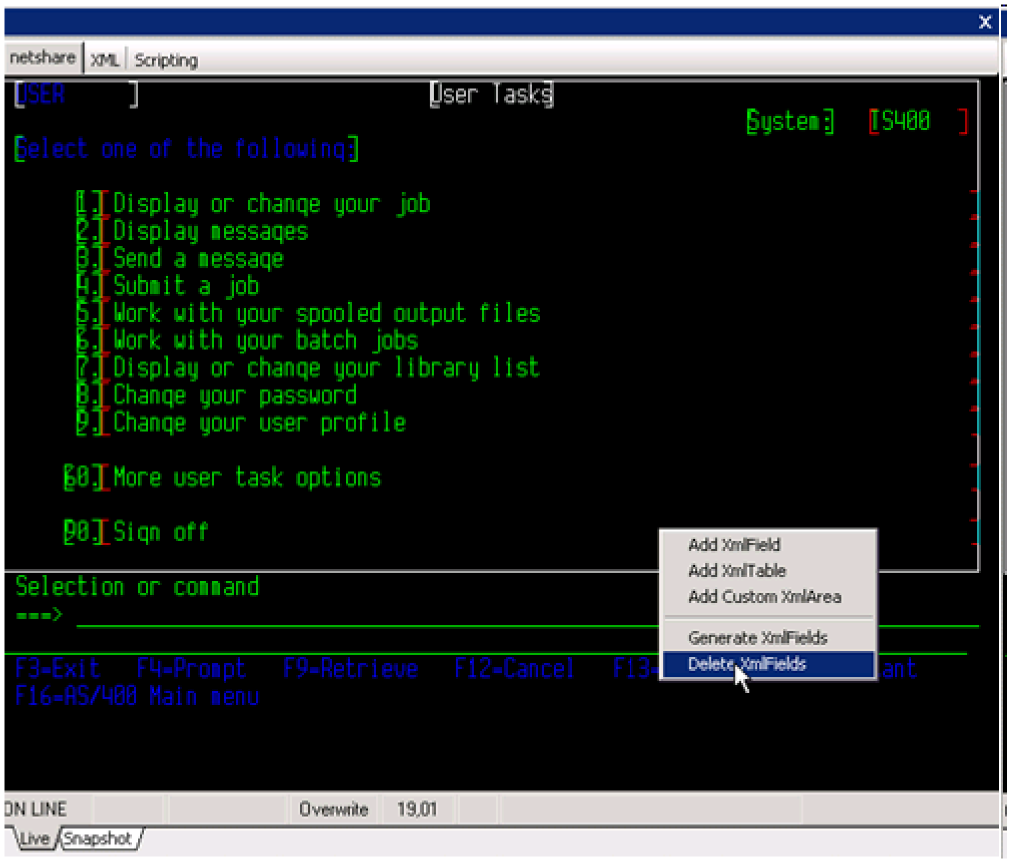
Once you have created and saved a couple of XML Template files you will notice that when you retrieve from host a screen that already has a XML Template that matchs, it is automatically selected and shown in the display. When a screen arrives, the XML Template engine looks in the template's directory 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.

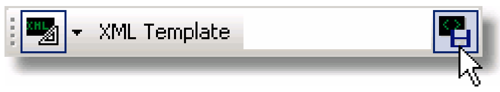


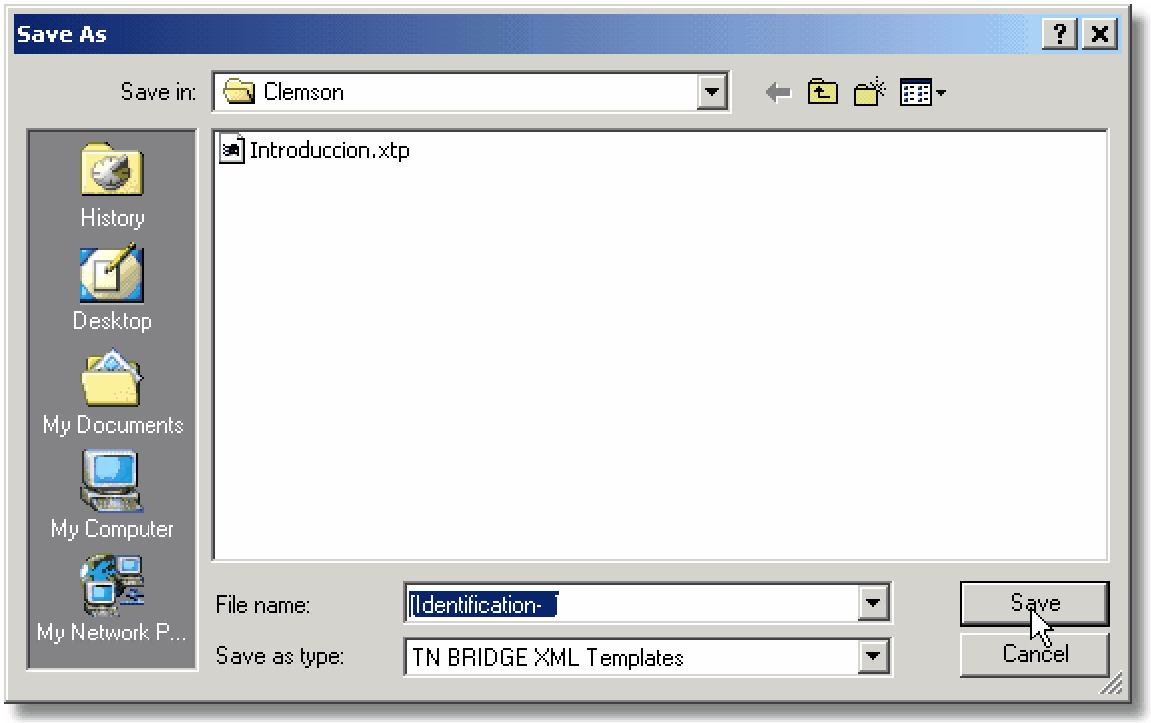
```
USER                              User Tasks
                                                        System:   TS400
Select one of the following:

     1. Display or change your job
     2. Display messages
     3. Send a message
     4. Submit a job
     5. Work with your spooled output files
     6. Work with your batch jobs
     7. Display or change your library list
     8. Change your password
     9. Change your user profile

    60. More user task options

    90. Sign off

Selection or command
===>

F3=Exit   F4=Prompt   F9=Retrieve   F12=Cancel   F13=            ant
F16=AS/400 Main menu
```

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.

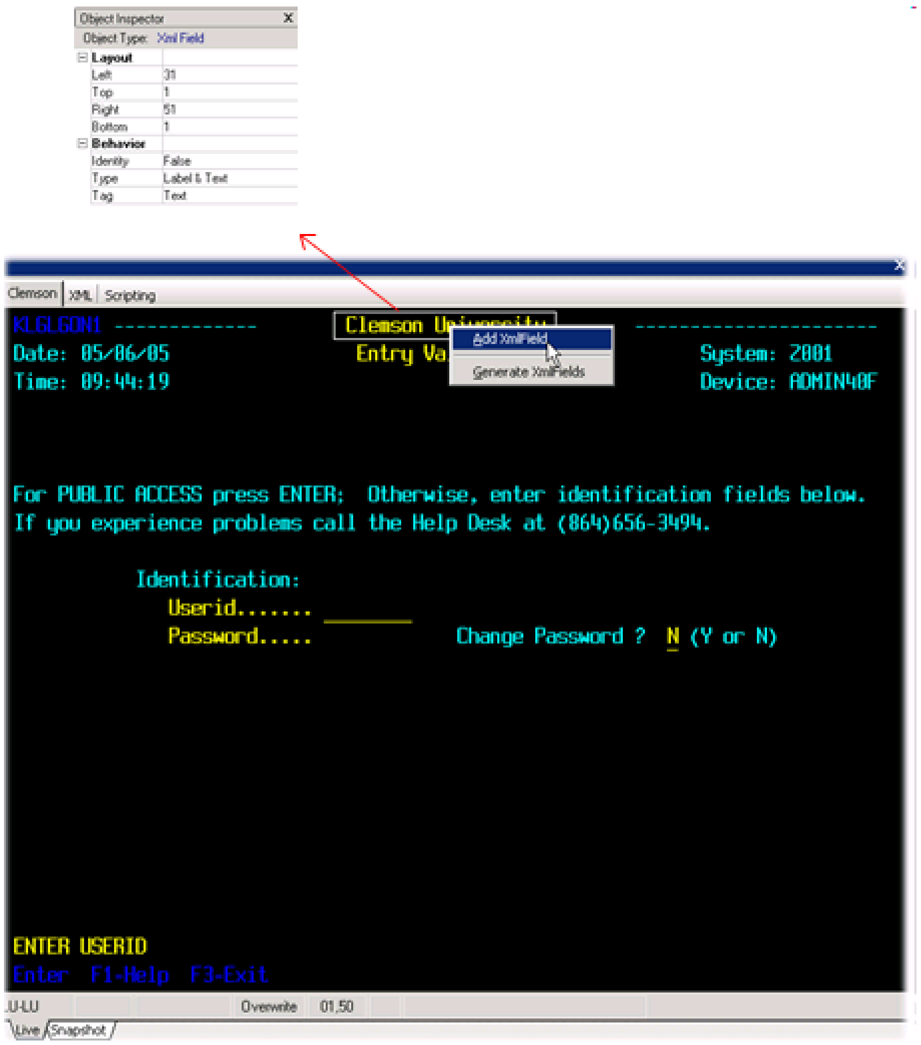




Note: Sometimes the save icon can be disabled, this is because the XML Template is empty. You must add some field before you can save it.

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**



Object Type: Xml Field

Layout	
Left	31
Top	1
Right	51
Bottom	1
Behavior	
Identity	False
Type	Label & Text
Tag	Text

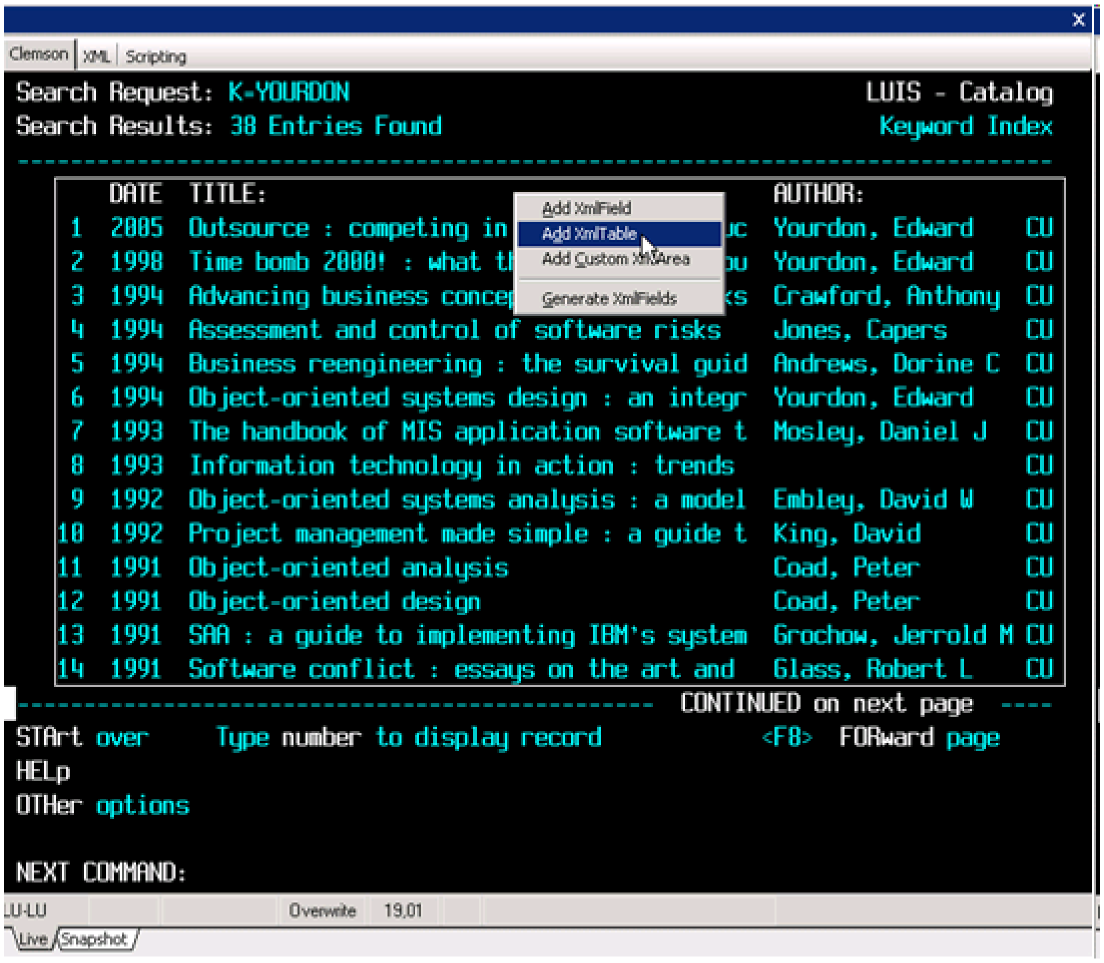
```
KLGLGON1 -------------       Clemson University       ----------------------
Date: 05/06/05                Entry Va                      System: 2001
Time: 09:44:19                                              Device: ADMIN40F


For PUBLIC ACCESS press ENTER;  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

           Identification:
              Userid....... ________
              Password.....              Change Password ?  N (Y or N)









ENTER USERID
Enter  F1-Help  F3-Exit
```

```
U-LU                    Overwrite   01,50
```

2.2.3.2.2.7 Adding XML Template Table

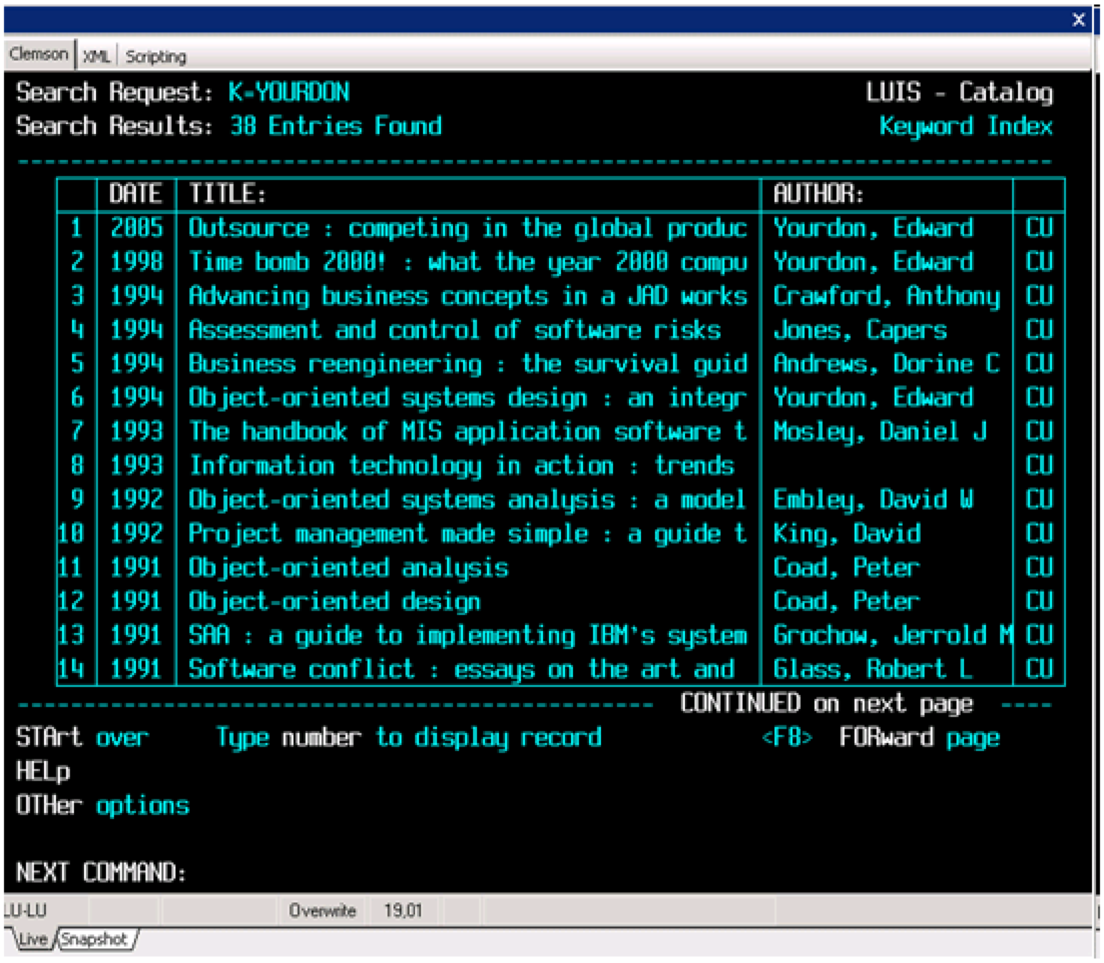
You can also create XML Template Tables.

- Select the area on which you want to create the table and right click on it.



```
Search Request: K-YOURDON                                       LUIS - Catalog
Search Results: 38 Entries Found                                 Keyword Index
-----------------------------------------------------------------------------
      DATE  TITLE:                                       AUTHOR:
   1  2005  Outsource : competing in                   uc Yourdon, Edward    CU
   2  1998  Time bomb 2000! : what th                  ou Yourdon, Edward    CU
   3  1994  Advancing business concep                  ks Crawford, Anthony  CU
   4  1994  Assessment and control of software risks     Jones, Capers       CU
   5  1994  Business reengineering : the survival guid   Andrews, Dorine C   CU
   6  1994  Object-oriented systems design : an integr   Yourdon, Edward     CU
   7  1993  The handbook of MIS application software t   Mosley, Daniel J    CU
   8  1993  Information technology in action : trends                        CU
   9  1992  Object-oriented systems analysis : a model   Embley, David W     CU
  10  1992  Project management made simple : a guide t   King, David         CU
  11  1991  Object-oriented analysis                     Coad, Peter         CU
  12  1991  Object-oriented design                       Coad, Peter         CU
  13  1991  SAA : a guide to implementing IBM's system   Grochow, Jerrold M  CU
  14  1991  Software conflict : essays on the art and    Glass, Robert L     CU
----------------------------------------------- CONTINUED on next page ----
STArt over       Type number to display record          <F8>  FORward page
HELp
OTHer options

NEXT COMMAND:
```

- The generated table will look like this.



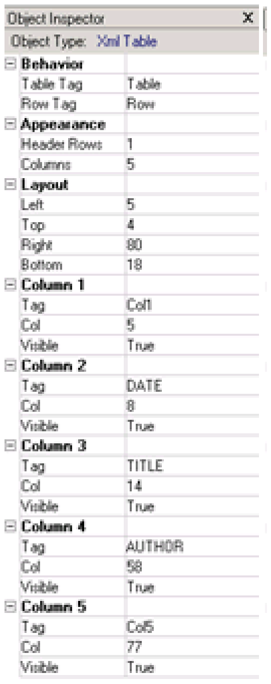
```
Search Request: K-YOURDON                                        LUIS - Catalog
Search Results: 38 Entries Found                                  Keyword Index
-----------------------------------------------------------------------------
```

	DATE	TITLE:	AUTHOR:	
1	2005	Outsource : competing in the global produc	Yourdon, Edward	CU
2	1998	Time bomb 2000! : what the year 2000 compu	Yourdon, Edward	CU
3	1994	Advancing business concepts in a JAD works	Crawford, Anthony	CU
4	1994	Assessment and control of software risks	Jones, Capers	CU
5	1994	Business reengineering : the survival guid	Andrews, Dorine C	CU
6	1994	Object-oriented systems design : an integr	Yourdon, Edward	CU
7	1993	The handbook of MIS application software t	Mosley, Daniel J	CU
8	1993	Information technology in action : trends		CU
9	1992	Object-oriented systems analysis : a model	Embley, David W	CU
10	1992	Project management made simple : a guide t	King, David	CU
11	1991	Object-oriented analysis	Coad, Peter	CU
12	1991	Object-oriented design	Coad, Peter	CU
13	1991	SAA : a guide to implementing IBM's system	Grochow, Jerrold M	CU
14	1991	Software conflict : essays on the art and	Glass, Robert L	CU

```
------------------------------------------------  CONTINUED on next page  ----
STArt over      Type number to display record          <F8>  FORward page
HELp
OTHer options

NEXT COMMAND:
```

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**.



Object Inspector

Object Type: Xml Table

Behavior	
Table Tag	Table
Row Tag	Row
Appearance	
Header Rows	1
Columns	5
Layout	
Left	5
Top	4
Right	80
Bottom	18
Column 1	
Tag	Col1
Col	5
Visible	True
Column 2	
Tag	DATE
Col	8
Visible	True
Column 3	
Tag	TITLE
Col	14
Visible	True
Column 4	
Tag	AUTHOR
Col	58
Visible	True
Column 5	
Tag	Col5
Col	77
Visible	True

Deleting the Table

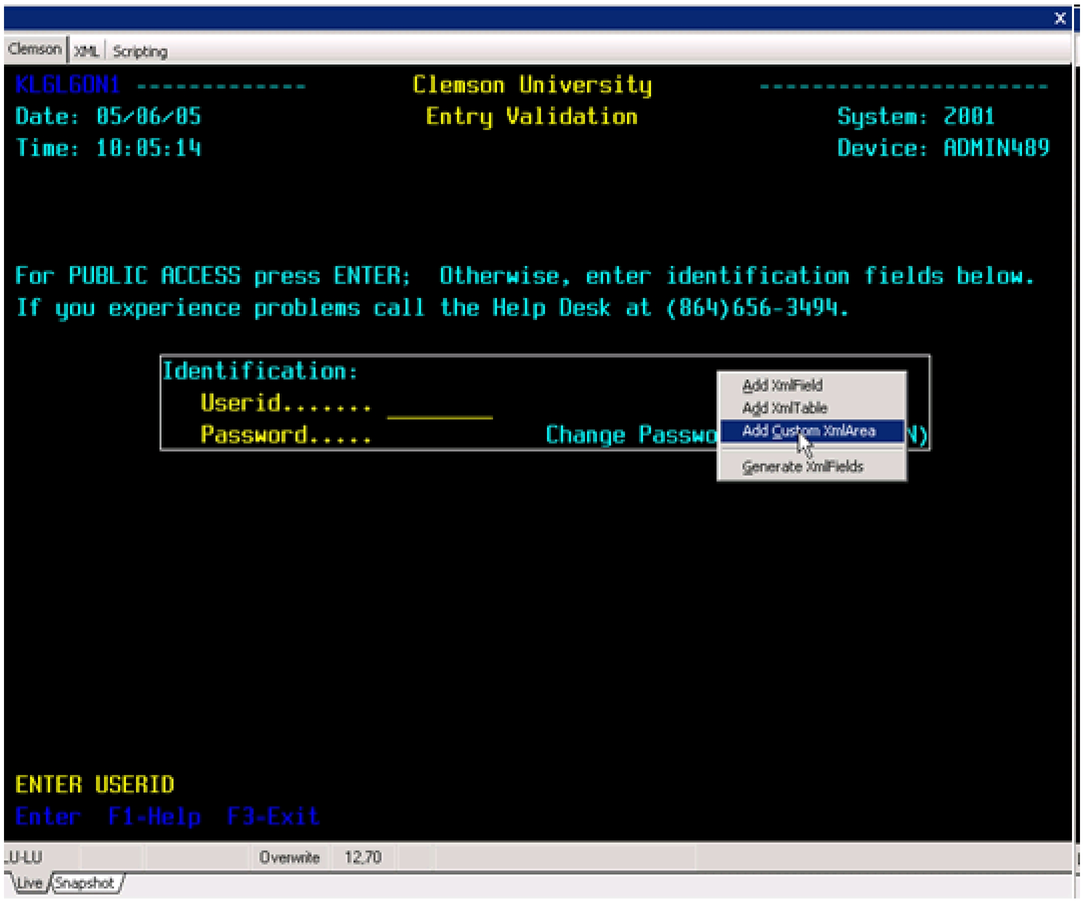
To delete the Table you have to do the inverse; by performing this action the XML Template will automatically be updated.

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.



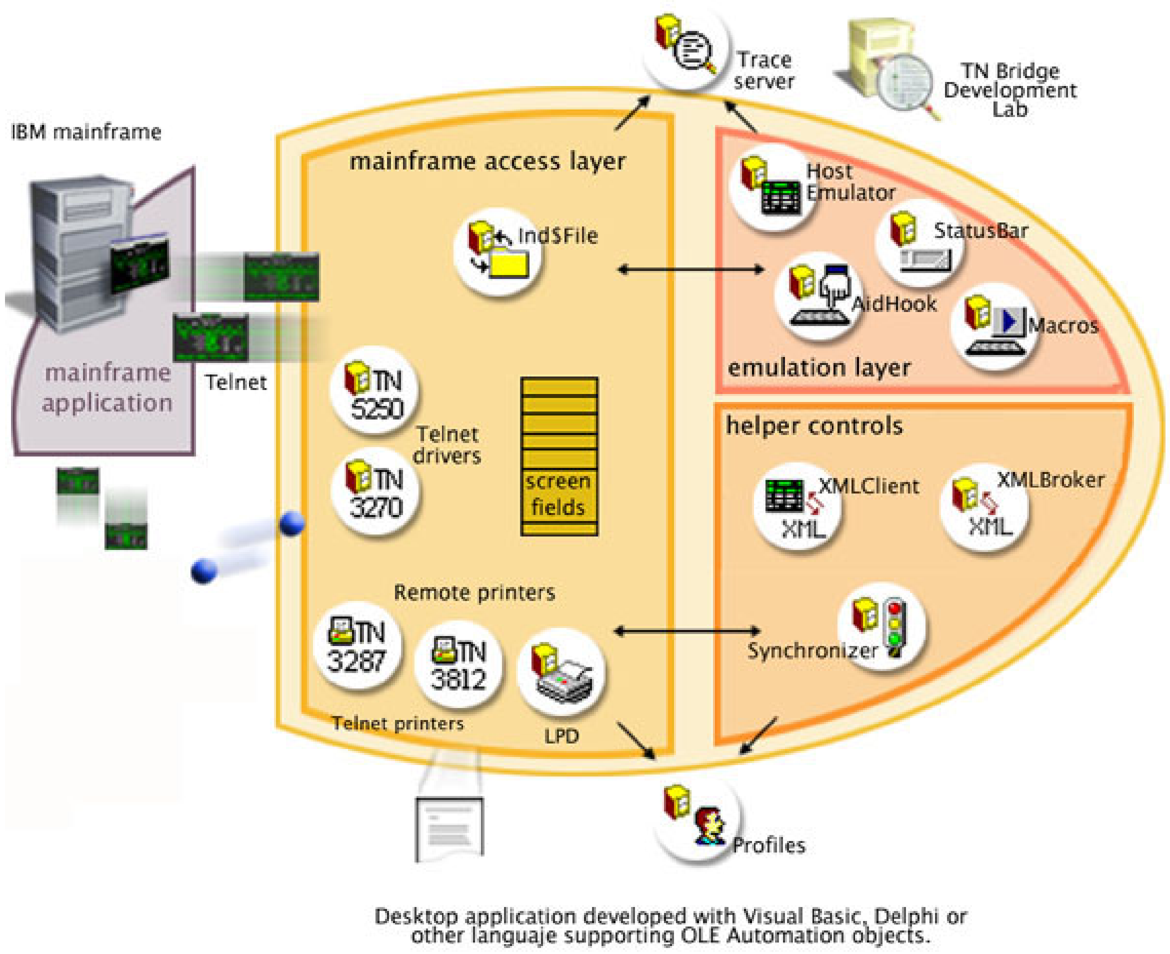
```
KLGLGON1 -------------         Clemson University        ----------------------
Date: 05/06/05                  Entry Validation                 System: 2001
Time: 10:05:14                                                   Device: ADMIN489



For PUBLIC ACCESS press ENTER;  Otherwise, enter identification fields below.
If you experience problems call the Help Desk at (864)656-3494.

            Identification:
               Userid.......  ________
               Password.....                Change Passwo           N)










ENTER USERID
Enter  F1-Help  F3-Exit
```

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 synchronic and asynchronic 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.

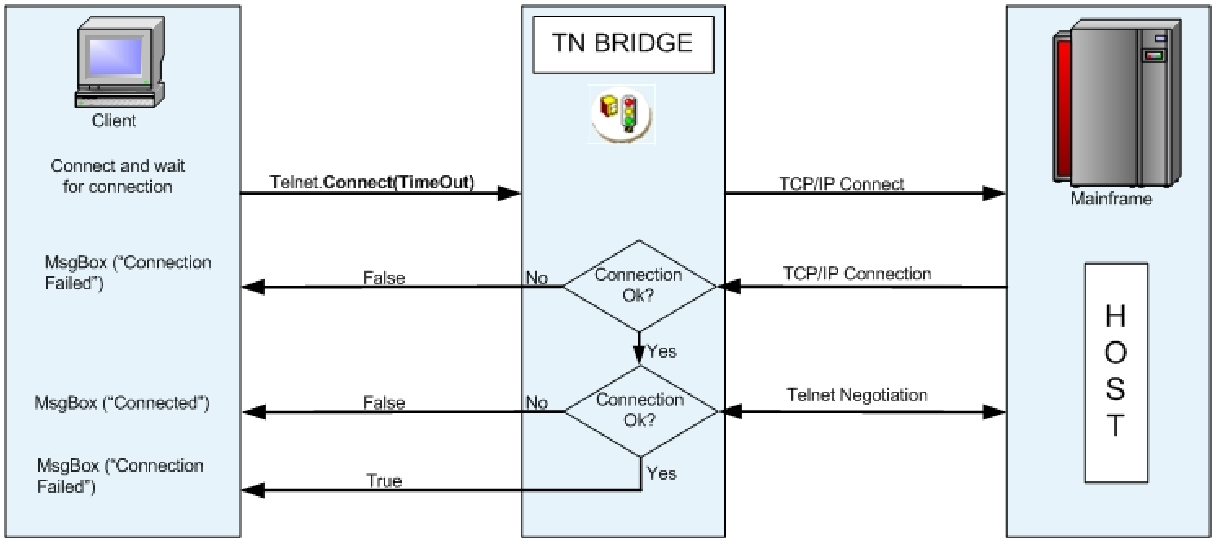
The Host and Port properties must be set for the connection to be successfully established.

VB

[Boolean] = *tnxxxx.***Connect(**Timeout As Integer**)**

Delphi
[Boolean] := *tnxxxx.***Connect(**Timeout:Integer**);**

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



Examples:

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

```
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 Asynchornous Connection

This method try to establish the connection to the specified Host and Port. This call **doesn't** block the code execution, so you should use the OnConnect and

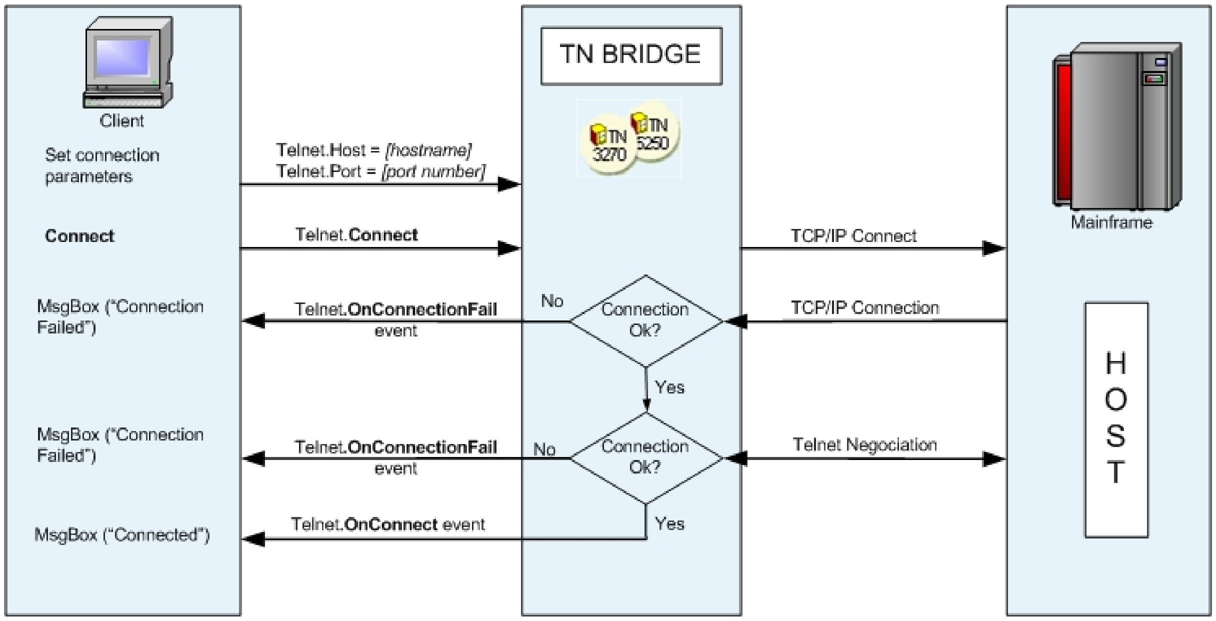
OnConnectionFail events to determine whether the connection is succesfully established or not. Alternativally, you can make a call to WaitForConnect synchronous method.

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

VB
*tnxxxx.***Connect()**

Delphi
*tnxxxx.***Connect;**

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



Examples:

VB:

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

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

Private Sub telnet_OnConnectionFail()
  MsgBox("Connection failed")
End Sub
```

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;
```

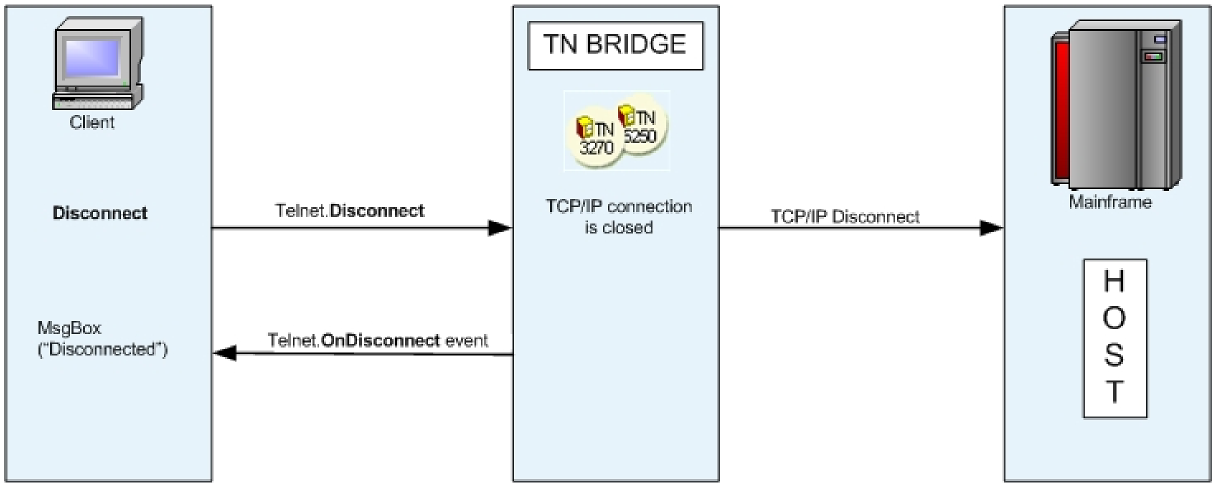
3.2.1.1.3 Disconnection

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:



Examples:

VB:

```
Private Sub cmdDisconnect_Click()
```

```
  telnet.Disconnect()
End Sub
```

Delphi:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  tnb3270.Disconnect;
end;
```

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
[*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
[*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:

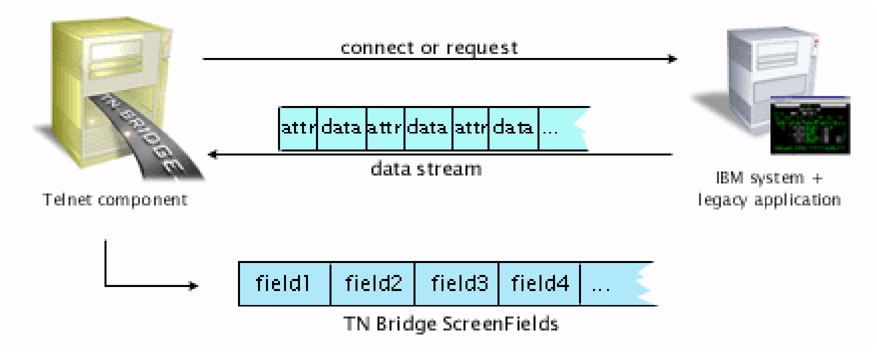
```
Private Sub telnet_OnScreenChange()
   ' this code gets the Text from position 150
   Dim s As String = telnet.GetText(150)
   ...
   ' this code gets the Text from position 150 to 200
   Dim s As String = telnet.GetText(150,200)
   ...
   ' this code gets the Text from row 15 column 4 with a lenght 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 lenght 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: *R[row number]C[column number]* For example, the field at row 4 column 30 in the screen is identified as *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

Sendind data to the mainframe can be splitted 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** explain 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.

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field

@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

Examples:

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

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
[*Field* =] *tnxxxx.***EditFields (***Index As Integer***)**
[*Field* =] *tnxxxx.***EditFields (***Index As String***)**

Delphi
[*Field* :=] *tnxxxx.***EditFields (***Index : Integer***);**
[*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
   telnet.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.

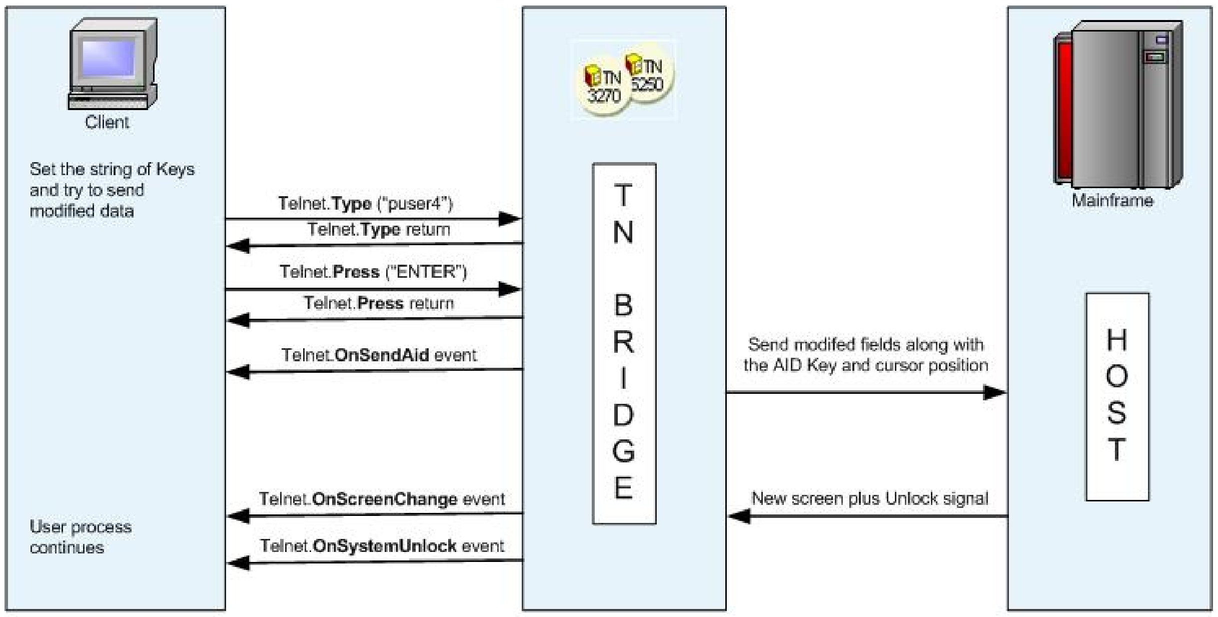
3.2.1.3.3.1 The Press method

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
*tnxxxx.***Press (***aidKey as AidKey***)**
*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:



Client
Set the string of Keys and try to send modified data
User process continues
TN 3270
TN 5250
T N B R I D G E
Mainframe
H O S T
Telnet.Type ("puser4")
Telnet.Type return
Telnet.Press ("ENTER")
Telnet.Press return
Telnet.OnSendAid event
Send modifed fields along with the AID Key and cursor position
New screen plus Unlock signal
Telnet.OnScreenChange event
Telnet.OnSystemUnlock event

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
   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
   telnet.Press("Enter")
End Sub
```

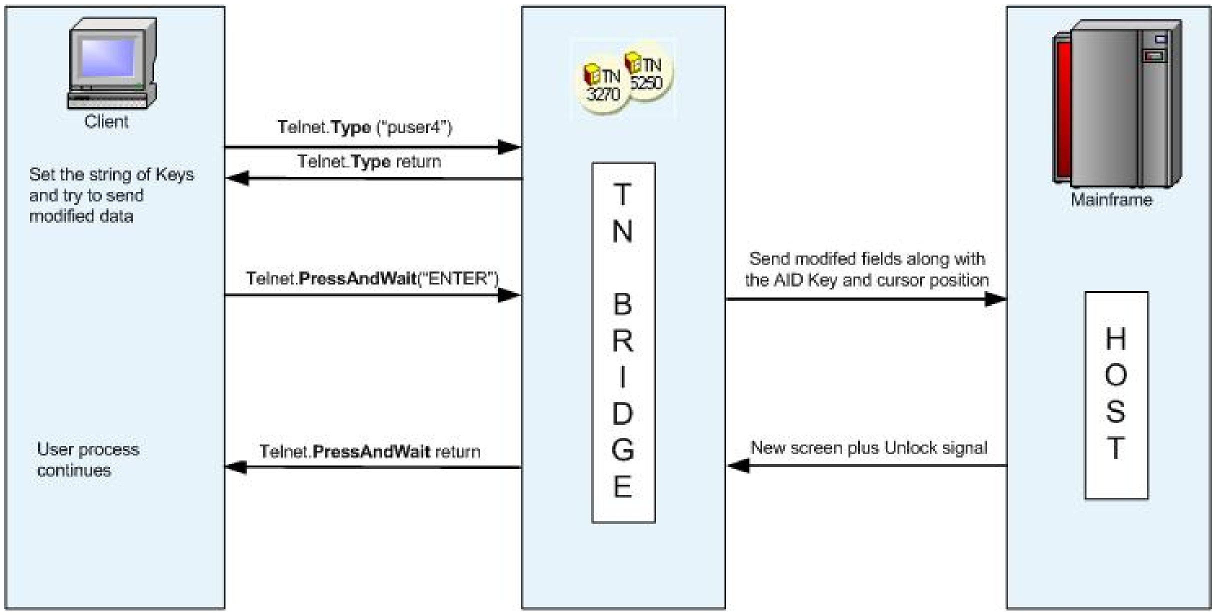
Delphi:
```
procedure TForm1.LoginClick(Sender:TObject);
begin
   // this code sets userId in Field located at Row 10 Column 2
   tnb3270.EditFields['R10C2'].Data := 'puser';
   // this code sets the password in Field located at Row 12 Column 2
   tnb3270.EditFields['R12C2'].Data := 'k2156';
   // this 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
*tnxxxx.***PressAndWait (***aidKey as AidKey***)**
*tnxxxx.***PressAndWait (***aidKey As String***)**

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


Client
Set the string of Keys and try to send modified data
User process continues
TN 3270
TN 5250
TN BRIDGE
Mainframe
HOST
Telnet.Type ("puser4")
Telnet.Type return
Telnet.PressAndWait("ENTER")
Telnet.PressAndWait return
Send modifed fields along with the AID Key and cursor position
New screen plus Unlock signal

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
   telnet.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;
```

3.2.1.4 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:
```
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:
```
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:
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:
procedure TForm1.Tnb3270E1ScreenChange(Sender: TObject);
begin
  ShowMessage('A new screen has been received');
end;

procedure TForm1.Tnb3270E1SystemUnLock(Sender: TObject);
begin
  ShowMessage('The latest screen has arrived');
end;
```

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

```
VB:
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:
procedure TForm1.Tnb3270E1SystemUnLock(Sender: TObject);
begin
  ShowMessage('We can send data to the host now');
end;

procedure TForm1.Tnb3270E1SystemLock(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:
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 = "MyPasword"
        '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 := 'MyPasword';
    // 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:

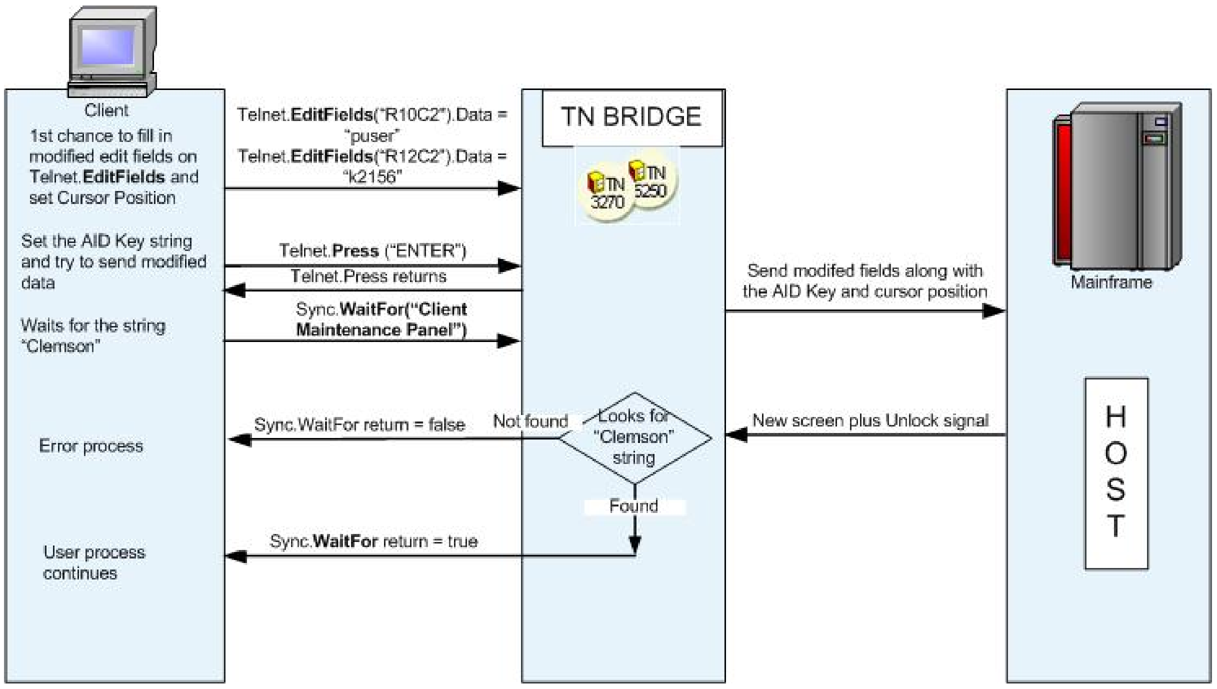
VB
[Boolean] = *tnxxxx.***WaitForConnect (***[TimeOut As Integer]***)**
[Boolean] = *tnxxxx.***WaitForScreen (***[TimeOut As Integer]***)**
[Boolean] = *tnxxxx.***WaitForUnlock (***[TimeOut As Integer]***)**
[Boolean] = *tnxxxx.***WaitFor (***StrValue As String, [TimeOut As Integer]***)**
[Integer] = *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 (***StrValue : 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:


Client
1st chance to fill in modified edit fields on Telnet.EditFields and set Cursor Position
Set the AID Key string and try to send modified data
Waits for the string "Clemson"
Error process
User process continues
Telnet.EditFields("R10C2").Data = "puser"
Telnet.EditFields("R12C2").Data = "k2156"
Telnet.Press ("ENTER")
Telnet.Press returns
Sync.WaitFor("Client Maintenance Panel")
Sync.WaitFor return = false
Not found
Sync.WaitFor return = true
TN BRIDGE
TN 3270
TN 5250
Looks for "Clemson" string
Found
Send modifed fields along with the AID Key and cursor position
New screen plus Unlock signal
Mainframe
HOST

Examples:

VB:
```
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:
```
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');
```

```
        Edit3.Text := TNB32701.GetText(20,5,30);
        TNB32701.Disconnect();
      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
tnbxxxx = *tnbxpool.***Acquire(**[sessionId as String]**)**

Delphi
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
*tnbxpool.***Release(tnobject)**

Delphi
*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:
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 transacion (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:
Private Sub Load()
  Dim Pool As TTNBXPOOL
  Dim telnet As TTNB3270X

  Set Pool = new TTNBXPOOL
  Pool.Id = "Clemson"
  Pool.Timout = 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:
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 appropiate 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:

```
<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 propery 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:

```
<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">
                              <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>
```

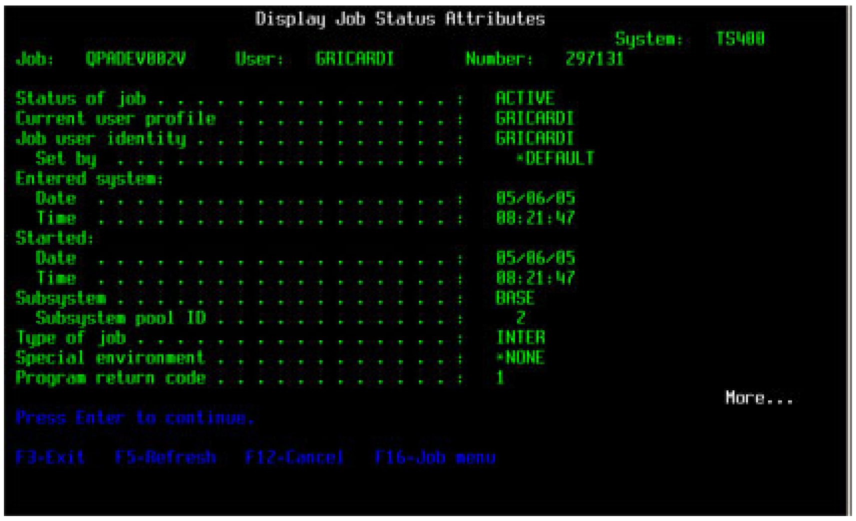
3.2.1.7.2 Providing a more friendly XML

Generating a more friendly XML involves the use of XMLTemplates. 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:

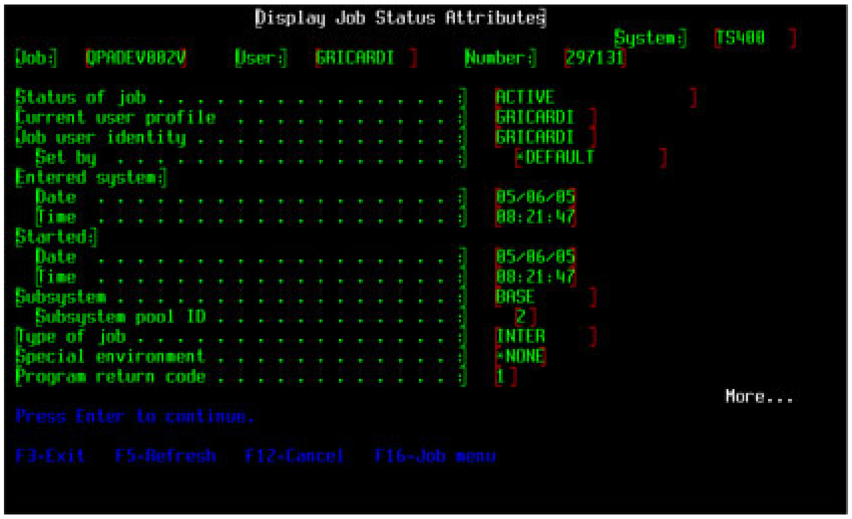


```
                    Display Job Status Attributes
                                                       System:   TS400
 Job:   QPADEV002V     User:   GRICARDI       Number:   297131

 Status of job . . . . . . . . . . . . . . . :   ACTIVE
 Current user profile  . . . . . . . . . . . :   GRICARDI
 Job user identity . . . . . . . . . . . . . :   GRICARDI
   Set by  . . . . . . . . . . . . . . . . . :     *DEFAULT
 Entered system:
   Date  . . . . . . . . . . . . . . . . . . :   05/06/05
   Time  . . . . . . . . . . . . . . . . . . :   08:21:47
 Started:
   Date  . . . . . . . . . . . . . . . . . . :   05/06/05
   Time  . . . . . . . . . . . . . . . . . . :   08:21:47
 Subsystem . . . . . . . . . . . . . . . . . :   BASE
   Subsystem pool ID . . . . . . . . . . . . :     2
 Type of job . . . . . . . . . . . . . . . . :   INTER
 Special environment . . . . . . . . . . . . :   *NONE
 Program return code . . . . . . . . . . . . :   1
                                                                  More...
 Press Enter to continue.

 F3=Exit   F5=Refresh   F12=Cancel   F16=Job menu
```

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.



```
                         Display Job Status Attributes
                                                         System:   TS400
Job:   QPADEV002V     User:   GRICARDI      Number:   297131

Status of job . . . . . . . . . . . . . . . :   ACTIVE
Current user profile  . . . . . . . . . . . :   GRICARDI
Job user identity . . . . . . . . . . . . . :   GRICARDI
  Set by  . . . . . . . . . . . . . . . . . :     *DEFAULT
Entered system:
  Date  . . . . . . . . . . . . . . . . . . :   05/06/05
  Time  . . . . . . . . . . . . . . . . . . :   08:21:47
Started:
  Date  . . . . . . . . . . . . . . . . . . :   05/06/05
  Time  . . . . . . . . . . . . . . . . . . :   08:21:47
Subsystem . . . . . . . . . . . . . . . . . :   BASE
  Subsystem pool ID . . . . . . . . . . . . :     2
Type of job . . . . . . . . . . . . . . . . :   INTER
Special environment . . . . . . . . . . . . :   *NONE
Program return code . . . . . . . . . . . . :   1
                                                                     More...
Press Enter to continue.

F3=Exit   F5=Refresh   F12=Cancel   F16=Job menu
```

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

```
<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
        <SetBy>*DEFAULT</SetBy>
  </JobUserIdentity>
  <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
        <SubsystemPoolID>2</SubsystemPoolID>
  </Subsystem>
  <TypeOfJob>INTER</TypeOfJob>
  <SpecialEnvironment>*NONE</SpecialEnvironment>
  <ProgramReturnCode>1</ProgramReturnCode>
```

</Screen>

It mightn't be perfect, but with an additional customization it might!

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 directroy. 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 matchs 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:
```
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** [= ***TNBXAid***]

Delphi Syntax

*TNBnnnnX.***AIDKey** [:= ***TNBXAid***];

It can take any of TNBAidKey constant values:

Constant Value	Meaning
akNone	No Action
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 **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:

Constant Value	Meaning
cpAustralia = 037	Australian
cpBelgium = 037	Belgium
cpCanada = 037	Canada
cpCzechRepublic = 870	Czech Republic
cpDenmark = 277	Denmark
cpFinland = 278	Finland
cpFrance = 297	France
cpGermany = 273	Germany

cpGreece = 423	Greece
cpGreece2 = 875	Greece
cpItaly = 280	Italy
cpLatinAmerica = 284	Latin America
cpNetherlands = 037	Netherlands
cpNorway = 277	Norway
cpPoland = 870	Poland
cpPortugal = 037	Portugal
cpSpain = 284	Spain
spSweden = 278	Sweden
cpSwitzerlandFrench = 500	Switzerland (French)
cpSwitzerlandGerman = 500	Switzerland (German)
cpUnitedKingdom = 285	United Kingdom
cpUnitedStates = 037	United States
cpInternational = 500	International

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

[*Integer* =] TNBnnnnX.Cols

Delphi Syntax

[*Integer* :=] TNBnnnnX.Cols;

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

*TNBnnnnX.***ConnectionName** [= *String*]

Delphi Syntax

*TNBnnnnX.***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 **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:

Constant Value	Meaning
True	Enables debug information.
False	Disables debug information.

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

*TNBnnnnX.***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.

4.1.2.1.2.9 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.***Extended** [= *Boolean*]

Delphi Syntax

*TNBnnnnX.***Extended** [:= *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 **Host**, **Port** and **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** [= *String*]

Delphi Syntax

*TNBnnnnX.***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.

4.1.2.1.2.13 HostFields

Array containing the screen fields.

Visual Basic Syntax

[*TNBXField* =] *TNBnnnnX.***HostFields (***Index As Variant***)**

Delphi Syntax

[*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

[*Boolean* =] *TNBnnnnX.***IsExtended**

Delphi Syntax

[*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

*TNBnnnnX.***KeepAlive** [= *Boolean*]

Delphi Syntax

*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

[*String* =] *TNBnnnnX.***LastErrMsg**

Delphi Syntax

[*String* :=] *TNBnnnnX.***LastErrMsg**;

4.1.2.1.2.17 Password

Sets a Password for the UserId specified.

Visual Basic Syntax

*TNBnnnnX.***Password** [= *String*]

Delphi Syntax

*TNBnnnnX.***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

*TNBXEmulator.***ProfilesControl** [= *TNBXProfiles*]

Delphi Syntax

*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

*TNBnnnnX.*Port [= *Integer*]

Delphi Syntax

*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

[*Integer* =] *TNBnnnnX*.Rows

Delphi Syntax

[*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

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

Delphi Syntax

[*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

[*String* =] *TNBnnnnX.***SessionId**

See Also **TNBXPOOL** class.

4.1.2.1.2.23 SessionState

Returns the current session state.

Visual Basic Syntax

[*TNBXSS* =] *TNBnnnnX*.**SessionState**

Delphi Syntax

[*TNBXSS* :=] *TNBnnnnX*.**SessionState**;

It can return any of TNBSessionState constant values:

Constant Value	Meaning
ssNoSession	Not in session.
ssSSCPLU	In session with VTAM.
ssLULU	In session with VTAM Application.

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

TNBnnnnX.**SSL** [= *Boolean*]

Delphi Syntax

TNBnnnnX.**SSL** [:= *Boolean*];

See also SSLMethod, SSLDisplay, SSLCertFile properties.

4.1.2.1.2.25 SSLAcceptExpired

Allows to accept or not expired certificates.

Visual Basic Syntax

*TNBnnnnX.***SSLAcceptExpired** [= *Boolean*]

Delphi Syntax

*TNBnnnnX.***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

*TNBnnnnX.***SSLAcceptInvalid** [= *Boolean*]

Delphi Syntax

*TNBnnnnX.***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

*TNBnnnnX.***SSLAcceptInvalidCA** [= *Boolean*]

Delphi Syntax

*TNBnnnnX.***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, SSLAcceptNotYetValid 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

*TNBnnnnX.***SSLDisplay** [= *Boolean*]

Delphi Syntax

*TNBnnnnX.***SSLDisplay** [:= *Boolean*];

See also SSL property.

4.1.2.1.2.33 SSLEnableDialogs

Allows to enable SSL dialogs mechanism.

Visual Basic Syntax

*TNBnnnnX.***SSLEnableDialogs** [= *Boolean*]

Delphi Syntax

*TNBnnnnX.***SSLEnableDialogs** [:= *Boolean*];

4.1.2.1.2.34 SSLKeyFile

Sets/gets the SSL key file.

Visual Basic Syntax

*TNBnnnnX.***SSLKeyFile** [= *String*]

Delphi Syntax

*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

*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:

Constant Value	Meaning
tt5211m2 = 0	IBM-5211-2 24 rows x 80 columns
tt3179m2 = 1	IBM-3179-2 24 rows x 80 columns
tt3477mFC = 2	IBM-3477-FC 27 rows x 132 columns
tt3180m2 = 3	IBM-3180-2 27 rows x 132 columns

Visual Basic Syntax

*TNBnnnnX.***TerminalType** [= **TNBX3270TT**]

Delphi Syntax

*TNBnnnnX.***TerminalType** [= **TNBX3270TT**];

For 3270:

Constant Value	Meaning
tt3278m2 = 0	IBM-3278-2 24 rows x 80 columns
tt3278m2E = 1	IBM-3278-2-E 24 rows x 80 columns extended
tt3278m3 = 2	IBM-3278-3 32 rows x 80 columns
tt3278m3E = 3	IBM-3278-3-E 32 rows x 80 columns extended
tt3278m4 = 4	IBM-3278-4 43 rows x 80 columns
tt3278m4E = 5	IBM-3278-4-E 43 rows x 80 columns extended

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 (**Apply As Boolean)

Delphi Syntax

*TNBnnnnX.***CloseEditor (**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:

String Value	Meaning
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.

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.

Code Value	**Meaning**
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3

@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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

- OnConnect
- OnConnectFail
- OnDataAvailable
- OnDisconnect
- OnEndDocument
- OnStartDocument

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:

Constant Value	Meaning
amDirect	Enables direct association between sessions.
amAssociate	The host handle the association between sessions.

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

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

Tnb3287X.**ConversionTable** [= *String*]

Delphi Syntax

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:

Constant Value	Meaning
True	Enables debug information.
False	Disables debug information.

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

*Tnb3287X.***DebugDir** [= *String*]

Delphi Syntax

*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 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

*Tnb3287X.***DeviceName** [= *String*]

Delphi Syntax

*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

Tnb3287X.**SendToPrinter**; [:= *Boolean*]

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

Visual Basic Syntax

*Tnb3287X.***SubKey** [= *String*]

Delphi Syntax

*Tnb3287X.***SubKey**; [:= *String*]

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

[*String* =] *Tnb3287X.***Text**

Delphi Syntax

[*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.

4.1.2.2.1.2 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**;

4.1.2.2.1.3 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**

- Connect
- Disconnect
- ShowPrinterEditor

Events

- OnConnect
- OnConnectFail
- OnDisconnect
- OnEndDocument
- OnDataAvailable
- OnStartDocument

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:

Constant Value	Meaning
cpAustralia = 037	Australian
cpBelgium = 037	Belgium
cpCanada = 037	Canada
cpCzechRepublic = 870	Czech Republic
cpDenmark = 277	Denmark
cpFinland = 278	Finland
cpFrance = 297	France
cpGermany = 273	Germany
cpGreece = 423	Greece
cpGreece2 = 875	Greece
cpItaly = 280	Italy
cpLatinAmerica = 284	Latin America
cpNetherlands = 037	Netherlands
cpNorway = 277	Norway
cpPoland = 870	Poland
cpPortugal = 037	Portugal
cpSpain = 284	Spain

spSweden = 278	Sweden
cpSwitzerlandFrench = 500	Switzerland (French)
cpSwitzerlandGerman = 500	Switzerland (German)
cpUnitedKingdom = 285	United Kingdom
cpUnitedStates = 037	United States
cpInternational = 500	International

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

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

*Tnb3812X.***Debug** [= *Boolean*]

Delphi Syntax

*Tnb3812X.***Debug**; [:= *Boolean*]

Remarks

The following are possible values for Debug property:

Constant Value	Meaning
True	Enables debug information.
False	Disables debug information.

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

*Tnb3812X.***DebugDir** [= *String*]

Delphi Syntax

*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 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

*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

[*Boolean* =] *Tnb3812X.***IsConnected**

Delphi Syntax

[*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

*Tnb3812X.***MsgQueueLib** [= *String*]

Delphi Syntax

*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

*Tnb3812X.***MsgQueueName** [= *String*]

Delphi Syntax

*Tnb3812X.***MsgQueueName**; [:= *String*]

See also **MsgQueueLib** property.

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 ammount of allocated obejcts 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 ammount of obejcts 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 ammount of allocated obejcts that are available.

Visual Basic Syntax

[*Integer* =] *TNBXPOOL.Count*

See also **Count** and **AcquiredCount** properties.

4.1.2.4.2.5 Size

Specifies the maximun ammount of objects for the pool.

Visual Basic Syntax

*TNBXPOOL.**Size*** [= *Integer*]

Remarks

When the pool size is exausted, 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

TNBXPOOL.Timeout [= *Integer*]

Remarks

Specifies the inactivity timeout for the pool. Once elapsed the specified time the timedout 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

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

*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

*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

*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

*TNBXField.***Attr** [= *Integer*]

Delphi Syntax

*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

[*Boolean* :=] *TNBXField.***Blinking**;

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

[*Boolean* :=] *TNBXField.***ByPass**;

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

Sets/gets 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

[*Boolean* =] *TNBXField.***MonoCase**

Delphi Syntax

[*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

*TNBXField.***Name** [= *String*]

Delphi Syntax

*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

[*Boolean* =] *TNBXField.***Numeric**

Delphi Syntax

[*Boolean* :=] *TNBXField.***Numeric**;

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

[*Boolean* :=] *TNBXField.***Visible**;

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 **TNBXField** 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

*TNBXFields.*Lock

Delphi Syntax

*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" witch 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_yyyymmddhhmmss.SYS : Contains trace and events messages
LPD_yyyymmddhhmmss.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:

State	Meaning
ldYes	The file is always deleted.
ldNo	The file is never deleted
ldClient	The file is deleted if the client demand to be deleted by a telnet command.

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

```
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

*Object.***Tag** [= *Integer*]

Delphi Syntax

*Object.***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

*TnbXLPD.***LoadFromFile (**FileName As String**)**

Delphi Syntax

*TnbXLPD.***LoadFromFile (**FileName:string**)**;

Example

*TnbXLPD.***LoadFromFile(**'C:\MyFolder\MyConfigFile.cfg'**)**

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

Visual Basic Syntax

*TnbXLPD.***SaveToFile (**FileName As String**)**

Delphi Syntax

*TnbXLPD.***SaveToFile (**FileName:string**)**;

Example

*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

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**)**

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 DataFileLenght 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

[*String* =] *TnbXLPDJob*.**ControlFileName**

Delphi Syntax

[*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

[*Integer* =] *TnbXLPDJob*.**ControlFileReceived**

Delphi Syntax

[*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

[*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

[*Boolean* :=] *TnbXLPDJob*.**IsConnected**;

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:

Constant Value	**Meaning**
pmText	Text mode
pmPostScript	PostScript mode

Visual Basic Syntax

[TXPrinterPrintMode =] *TnbXLPDJob*.**PrintMode**

Delphi Syntax

[TXPrinterPrintMode :=] *TnbXLPDJob*.**PrintMode**;

See also **Status** property.

4.1.2.8.2.12 Queue

Its 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:

Constant Value	Meaning
lsWaitCommand	Waiting a command from the host
lsWaitData	Receiving the data file
lsWaitControl	Receiving the control file
lsWaitSubcommand	Waiting a subcommand from the host
lsAbortJob	The job was aborted

Visual Basic Syntax

[TxTnbLPDStatus =] *TnbXLPDJob*.**Status**

Delphi Syntax

[TxTnbLPDStatus :=] *TnbXLPDJob*.**Status**;

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 **TnbXLPDJob** class.

Constant Value	**Meaning**
pmText	Text mode
pmPostScript	PostScript mode

4.1.2.8.3.2 TxTnbLPDStatus

These values are used in the **Status** property of **TnbXLPDJob** class.

Constant Value	**Meaning**
lsWaitCommand	Waiting a command from the host
lsWaitData	Receiving the data file
lsWaitControl	Receiving the control file
lsWaitSubcommand	Waiting a subcommand from the host
lsAbortJob	The job was aborted

4.1.2.9 TnbXLPDJobs Reference

4.1.2.9.1 TnbXLPDJobs 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
- ShowPrinterEditor

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

[*TnbXLPDQueue* :=] *TnbXLPDQueue*.**CurrentJob**;

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:

Constant Value	Meaning
ldYes	The file is always deleted
ldNo	The file never is deleted
ldClient	The file is deleted if the client demands to be deleted by a telnet command

Visual Basic Syntax

TnbXLPDQueue.**DeleteFileAfterPrint** [= TxTnbXLPDDelete]

Delphi Syntax

TnbXLPDQueue.**DeleteFileAfterPrint**; [:= TxTnbXLPDDelete]

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

TnbXLPDQueue.**Enabled** [= *Boolean*]

Delphi Syntax

TnbXLPDQueue.**Enabled**; [:= *Boolean*]

4.1.2.10.2.5 Jobs

Contains a collection of TnbXLPDJobs objects, one for each job in the queue.

Visual Basic Syntax

[*TnbXLPDJob* **=**] *TnbXLPDQueue*.**Jobs**

Delphi Syntax

[*TnbXLPDJob* :**=**] *TnbXLPDQueue*.**Jobs**;

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

[*TnbXLPD* =] *TnbXLPDQueue*.**LPD**

Delphi Syntax

[*TnbXLPD* :=] *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:

'*'	Can be replaced with any string
'?'	Can be replaced with any character

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

[*LongInt* :=] *TnbXLPDQueue.***GetControlHandle**;

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.

Constant Value	Meaning
ldYes	The file is always deleted
ldNo	The file never is deleted
ldClient	The file is deleted if the client demands to be deleted by a telnet command

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 strings 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 (**Lrecl** 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

TNBXIndFile.**BufSize** [= Integer]

Delphi Syntax

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, **HostFileName** in **Send** or **Run** (**Direction**=0) methods, or **LocalFileName** in **Receive** or **Run** (**Direction**=1) methods.

Visual Basic Syntax

TNBXIndFile.**Bytes** [Integer =]

Delphi Syntax

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:

Constant Value	Meaning
ftDFT	DFT mode
ftCut	Cut mode

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]

See also **ProfilesControl** property, **SubKey** property.

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:

Constant Value	Meaning
hVM	VM
hTSO	TSO
hCics	Cics

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

TNBXIndFile.**LocalFileName** [= FileName.dat]

Delphi Syntax

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

TNBXIndFile.**LRecl** [= Integer]

Delphi Syntax

TNBXIndFile.**LRecl**; [:= Integer]

See also **HostFileName** property, **BlkSize** property, **RecFm** property, **Run** method, **Send** method.

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, **SecSpace** property, **BlkSize** property, **RecFm** property, **Units** 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:

Value	Meaning
0	Default file format
1	Fixed file format
2	Variable file format
3	Undefined file format

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]

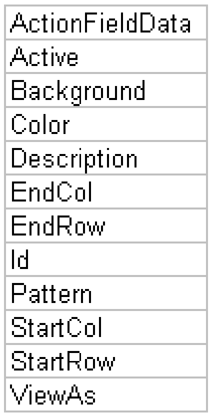
Delphi Syntax

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.



ActionFieldData
Active
Background
Color
Description
EndCol
EndRow
Id
Pattern
StartCol
StartRow
ViewAs

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.

Visual Basic Syntax

*TNBXIndFile.***TelnetControl** [= *TNBnnnnX*]

Delphi Syntax

*TNBXIndFile.***TelnetControl**; [:= *TNBnnnnX*]

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.**TimeOut** [= Integer]

Delphi Syntax

TNBXIndFile.**TimeOut**; [:= Integer]

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).

Value	**Meaning**
0	Default unit of allocation
1	Tracks unit of allocation
2	Cylinder unit of allocation
3	Average Blocks unit of allocation

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

TNBXIndFile.**AboutBox**

Delphi Syntax

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

TNBXIndFile.**Abort()**

Delphi Syntax

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

TNBXIndFile.**AsVariant()**

Delphi Syntax

TNBXIndFile.**AsVariant()**;

4.1.2.12.3.4 LoadFromXMLFile

Loads the XML code corresponding to the *TNBXHotSpots object.*

Visual Basic Syntax

*TNBXIndFile.***LoadFromXMLFile (***XMLFile As String***)**

Delphi Syntax

*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

TNBXIndFile.**Receive()**

Delphi Syntax

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

TNBXIndFile.**Run()**

Delphi Syntax

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

*TNBXIndFile.***SaveToXMLFile** ***(****XMLFile As String****)***

Delphi Syntax

*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

TNBXIndFile.**Send()**

Delphi Syntax

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 TxFtMode

These values are used in the **FTMode** property of **TnbXIndFile** control.

Constant Value	**Meaning**
ftDFT	DFT mode
ftCut	Cut mode

4.1.2.12.5.2 TxHostKind

These values are used in the **HostKind** property of **TNBXIndFile** control.

Constant Value	Meaning
hVM	VM
hTSO	TSO
hCics	Cics

4.1.2.13 Constants

4.1.2.13.1 TNBSSLMethod

These values are used for the SSLMethod property of TNB5250X control.

Constant Value	Meaning
metSSL23	Supports SSL 2.0 and 3.0 version.
metSSL2	Supports SSL 2.0 version.
metSSL3	Supports SSL 3.0 version.
metTLS1	Supports TLS 1.0 version.

4.1.2.13.2 TNBXCP

These values are used in the **CodePage** property of telnet controls.

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

cpUnitedStates = 037	United States
cpInternational = 500	International

4.1.2.13.3 TNBXAid

These values are used in the **AidKey** property of telnet controls.

Constant Value	Meaning
akNone	No Action
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

4.1.2.13.4 TNBXSS

These values are used in the **SessionState** property of telnet controls.

Constant Value	Meaning
ssNoSession	Not in session.

ssSSCPLU	In session with VTAM.
ssLULU	In session with VTAM Application.

4.1.2.13.5 TNBX5250TT

These values are used in the **TerminalType** property of TNBX5250 control.

Constant Value	**Meaning**
tt5211m2 = 0	IBM-5211-2 24 rows x 80 columns
tt3179m2 = 1	IBM-3179-2 24 rows x 80 columns
tt3477mFC = 2	IBM-3477-FC 27 rows x 132 columns
tt3180m2 = 3	IBM-3180-2 27 rows x 132 columns

4.1.2.13.6 TNBX3270TT

These values are used in the **TerminalType** property of TNBX3270 control.

Constant Value	**Meaning**
tt3278m2 = 0	IBM-3278-2 24 rows x 80 columns
tt3278m2E = 1	IBM-3278-2-E 24 rows x 80 columns extended
tt3278m3 = 2	IBM-3278-3 32 rows x 80 columns
tt3278m3E = 3	IBM-3278-3-E 32 rows x 80 columns extended
tt3278m4 = 4	IBM-3278-4 43 rows x 80 columns
tt3278m4E = 5	IBM-3278-4-E 43 rows x 80 columns extended

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

- **AbouBox**
- **AidKeyPressed**
- **AsVariant**
- ClassNameIs
- **Copy**
- **GetScreenRowEx**
- **GetScreenText**
- **LoadFromXMLFile**
- **LocalKeyPressed**
- **Paste**
- **PrintScreen**
- **PutFields**
- **SaveToXMLFile**
- **SendKeys**
- **ShowEditor**

Events

- **OnClick**
- **OnCursorPos**
- **OnDblClick**
- **OnInputInhibitChange**
- **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

*TNBXEmulator.***ErrorLine** [:= *Integer*];

Remarks

Default value is **0**.

See Also **DisableKbdLockOnError** property.

4.1.3.1.2.7 FontAutoSize

Set/gets 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.***InputInhibitEnabled** [= *Boolean*]

Delphi Syntax

*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

*TNBXEmulator.***InsertMode** [= *Boolean*]

Delphi Syntax

*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

*TNBXEmulator.***RulerType** [:= **TxRulerType**];

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:

Constant Value	Meaning
ruNone = 0	No lines
ruCrosshair = 1	Horizontal and vertical lines
ruVertical = 2	Only vertical line
ruHorizontal = 3	Only horizontal line

See Also **CursorPos** property and **TxRulerType** constants.

4.1.3.1.2.15 StyleName

Sets/gets the Style from the collection of styles saved through the **TNBXProfiles** ActiveX Control.

Visual Basic Syntax

*TNBXEmulator.***StyleName** [= *String*]

Delphi Syntax

*TNBXEmulator.***StyleName** [:= *String*];

Remarks

This property is valid if a **ProfilesControl** property has been previously assigned.

See Also **TNBXProfiles** control and **ShowEditor** method.

4.1.3.1.2.16 SubKey

Sets/gets the subkey under which the screen styles will be stored.

Visual Basic Syntax

*TNBXEmulator.***SubKey** [= *String*]

Delphi Syntax

*TNBXEmulator.***SubKey** [:= *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

*TNBXEmulator.***TelnetControl** [= *TNBnnnnX*]

Delphi Syntax

*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

*TNBXEmulator.***TopMargin** [= *Integer*]

Delphi Syntax

*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

*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**

Delphi 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],* *[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.

4.1.3.1.3.7 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

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

Delphi Syntax

[*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 GetScreenRowEx method.

4.1.3.1.3.8 LoadFromXMLFile

Loads the XML code corresponding to the *TNBXEmulator object.*

Visual Basic Syntax

*TNBXEmulator.*LoadFromXMLFile *(XMLFile As String)*

Delphi Syntax

*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

*TNBXEmulator.*LocalKeyPressed (*Value As String*)

Delphi Syntax

*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

*TNBXEmulator.*Paste

Delphi Syntax

*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.

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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.

Constant Value	Meaning
ruNone = 0	No lines
ruCrosshair = 1	Horizontal and vertical lines
ruVertical = 2	Only vertical line
ruHorizontal = 3	Only horizontal line

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

*TNBXAidHook.***KbdKind** [= **TNBKbdKind**]

Delphi Syntax

*TNBXAidHook.***KbdKind** [:= **TNBKbdKind**];

It can take one of this constants values:

Constant Value	Meaning
k5250	5250 Keyboard
k3270	3270 Keyboard

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

*TNBXAidHook.***Mapping3270** [= *KeyboardMap*]

Delphi Syntax

*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

*TNBXAidHook*_OnAidKey (ByVal Value As String)

Delphi Syntax

TNBXAidHook.OnAidKey (Sender: TObject)

Remarks

Value parameter can be one of the following strings:

Local processing function keys:

Value	Meaning
Up	go to previous row
Down	go to next row
Left	go to previous column
Right	go to next column
Insert	set the keyboard in inserting state
Delete	delete the next character
BackSpace	delete the previous character
ScreenBegin	go to row 1 column 1
ScreenEnd	go to last row last column
NextLine	go to column 1 of the next line
PriorField	go to the previous field
NextField	go to the next field
Home	go to the first field
End	go to last character of current field
EraseField	erase the current field
EraseEof	erase characters from current position until the end of field
FieldMinus	skip to the previous field
FieldPlus	skip to the next field
Restore	restore the current screen

Host processing function keys (Attention Identifier "AID" keys):

Value	Meaning
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** 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.

Constant Value	**Meaning**
k5250	5250 Keyboard
k3270	3270 Keyboard

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

*TNBXStatusBar.***EmulatorControl** [= *TNBEmulator*]

Delphi Syntax

*TNBXStatusBar.***EmulatorControl** [:= *TNBEmulator*];

Remarks

You must set this property for TNBXStatusBar control works properly.

4.1.3.3.2.2 Font

Sets/gets the font for the current connection.

Visual Basic Syntax

*TNBXStatusBar.***Font** [= *IFontDisp*]

Delphi Syntax

*TNBXStatusBar.***Font** [:= *IFontDisp*];

4.1.3.3.2.3 TelnetControl

Sets the TNB3270X or TNB5250X ActiveX Control as the telnet client control.

Visual Basic Syntax

*TNBXStatusBar.***TelnetControl** [= *TNBnnnnX*]

Delphi Syntax

*TNBXStatusBar.***TelnetControl** [:= *TNBnnnnX*];

Remarks

You must set this property for TNBStatusBar control to allow it to work properly.

4.1.3.3.2.4 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.

4.1.3.3.3 Methods

4.1.3.3.3.1 AboutBox

Shows Integration Pack about dialog box.

Visual Basic Syntax

*TNBXStatusBar.***AboutBox**

Delphi Syntax

*TNBXStatusBar.***AboutBox**;

4.1.3.3.3.2 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

*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

*TNBXMacro.***ProfilesControl** [= *TNBXProfiles*]

Delphi Syntax

*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

[*TNBMacroState* =] *TNBXMacro*.State

Delphi Syntax

[*TNBMacroState* :=] *TNBXMacro*.State;

Remarks

The value of this property can be one of the following constants:

Constant Value	Meaning
msNoMacro	No macro is loaded.
msStopped	The control has a macro loaded and is in a stopped state.
msRecording	The control is recording a macro.
msPlaying	The control is playing the macro loaded.

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.

Constant Value	Meaning
msNoMacro	No macro is loaded.
msStopped	The control has a macro loaded and is in a stopped state.
msRecording	The control is recording a macro.
msPlaying	The control is playing the macro loaded.

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

*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 and 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

[*Variant* =] *TNBXProfiles.***AsVariant**

Delphi Syntax

[*Variant* :=] *TNBXProfiles.***AsVariant**;

4.1.3.5.3.3 Commit

Writes the XML configuration to the profile.

Visual Basic Syntax

*TNBXProfiles.***Commit**

Delphi Syntax

*TNBXProfiles.***Commit**;

See also **Rollback** method.

4.1.3.5.3.4 Delete

Deletes the profile specified.

Visual Basic Syntax

*TNBXProfiles.***Delete (***SubKey As String, Profile As String***)**

Delphi Syntax

*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

*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

*TNBXProfiles.***Rollback**

Delphi Syntax

*TNBXProfiles.***Rollback**;

See also **Commit** method.

4.1.3.5.3.9 SetCurrent

Sets the current profile.

Visual Basic Syntax

*TNBXProfiles.***SetCurrent (***Source As String, CurrentProfile As String, id As String=' ', type_ As string=' '***)**

Delphi Syntax

*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

*TNBXHotSpot.***ActionFieldData** [= *String*]

Delphi Syntax

*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:

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:

Constant Value	Meaning
hsButton	Hotspot button style is displayed.
hsLink	Hotspot link style is displayed.
hsNone	No hotspot is displayed.
hsPlain	Hotspot plain text style is displayed.

4.1.3.6.3 Constants

4.1.3.6.3.1 TXTnbHotSpotViewAs

These values are used in the **ViewAs** property of hotspot control.

Constant Value	Meaning
hsButton	Hotspot button style is displayed.
hsLink	Hotspot link style is displayed.
hsNone	No hotspot is displayed.
hsPlain	Hotspot plain text style is displayed.

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

[*Integer* :=] *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

[*Integer* =] *TNBXHotSpots.***EndCol**

Delphi Syntax

[*Integer* :=] *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

[*Integer* =] *TNBXHotSpots.***EndRow**

Delphi Syntax

[*Integer* :=] *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:

Constant Value	Meaning
hsButton	Hotspot button style is displayed.
hsLink	Hotspot link style is displayed.
hsNone	No hotspot is displayed.
hsPlain	Hotspot plain text style is displayed.

4.1.3.7.3 Methods

4.1.3.7.3.1 AboutBox

Shows Integration Pack about dialog box.

Visual Basic Syntax

*TNBXHotSpots.***AboutBox**

Delphi Syntax

*TNBXHotSpots.***AboutBox**;

4.1.3.7.3.2 Add

Adds a new TnbXHotspot to the **TNBXHotSpots** collection.

Visual Basic Syntax

*[TNBXHotSpot =] TNBXHotSpots.***Add**

Delphi Syntax

*[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

[*Variant* =] *TNBXHotSpots.***AsVariant**

Delphi Syntax

[*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

*TNBXHotSpots.*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

*TNBXSync.***TelnetControl** [= *TNBnnnnX*]

Delphi Syntax

*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

*TNBXSync.***TraceControl** [= *TNBXTrace*]

Delphi Syntax

*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

*TNBXSync.***WaitTimeout** [= *Integer*]

Delphi Syntax

*TNBXSync.***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.

4.1.4.1.3 Methods

4.1.4.1.3.1 AboutBox

Shows Integration Pack about dialog box.

Visual Basic Syntax

*TNBXSync.***AboutBox**

Delphi Syntax

*TNBXSync.***AboutBox**;

4.1.4.1.3.2 AsVariant

Returns the control as a variant data type variable.

Visual Basic Syntax

[*Variant* =] *TNBXSync.***AsVariant**

Delphi Syntax

[*Variant* :=] *TNBXSync.***AsVariant**;

4.1.4.1.3.3 Connect

The Connect method establishes a synchronous connection to the Telnet server.

Visual Basic Syntax

[Boolean] = *TNBXSync.*Connect(*[TimeOut]*)

Delphi Syntax

[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

[Boolean] = *TNBXSync.*Disconnect(*[TimeOut] As Integer*)

Delphi Syntax

[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.

Visual Basic Syntax

[*Boolean* =] *TNBXSync*.**IsConnected**

Delphi Syntax

[*Boolean* :=] *TNBXSync*.**IsConnected**;

See also **Connect** and **Disconnect** methods.

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

TNBXSync.SendKeys (*Keys As String*)

Delphi Syntax

TNBXSync.SendKeys (*Keys:string*);

Remarks

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

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6

@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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 an screen containing the specified string.

Visual Basic Syntax

*[Boolean] = TNBXSync.*WaitFor(*StrValue As String, [TimeOut] As Integer*)

Delphi Syntax

*[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

*[Boolean] = TNBXSync.*WaitForConnect(*[TimeOut] As Integer*)

Delphi Syntax

*[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

[Boolean] = *TNBXSync.*WaitForStartDoc (*[TimeOut] As Integer*)

Delphi Syntax

[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

[Boolean] = *TNBXSync.*WaitForUnlock (*[TimeOut] As Integer*)

Delphi Syntax

[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 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

- **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

[*String* =] *XmlTemplate.***Id**

C# Syntax

[*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

[*Boolean* =] *XmlTemplate.***CanProduce**

C# Syntax

[*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

[*String* =] *XmlTemplate.***HostToXml**

C# Syntax

[*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

*XmlTemplate.***HostToXml**(Xml As String)

C# Syntax

*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

*TNBXXmlBroker.***XML** [= *String*]

Delphi Syntax

*TNBXXmlBroker.***XML** [*:=* *String*];

See also **XML Reference** for TnbXXmlBroker.

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:

```
<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 />
```

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 TnbXXmlBroker 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 propery 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:

```
<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">
                          <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 comunication 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:

Constant Value	Meaning
akNone	No Action
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 **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 necesary.

Visual Basic Syntax

*TNBXXmlClient.***AIDString** [= AidKey As String]

Delphi Syntax

*TNBXXmlClient.***AIDString** [:= AidKey:String];

The following table lists the possible string values:

String Value	Meaning
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 **SendAid** method.

4.1.4.4.2.3 ExcludeEmptyFields

Sets/gets the possibility of excluding all empty fields.

Visual Basic Syntax

*TNBXXmlClient.***ExcludeEmptyFields** [= Boolean]

Delphi Syntax

*TNBXXmlClient.***ExcludeEmptyFields** [:= Boolean];

4.1.4.4.2.4 HotSpotsControl

Sets/gets the **TNBXHotSpot** component as the hotspots handler for this control.

Visual Basic Syntax

*TNBXmlClient.***TNBXHotSpots** [= **TNBXHotSpot**]

Delphi Syntax

*TNBXmlClient.***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* =] *TNBXXmlClient.***ScreenRow (***Index As Integer***)**

Delphi Syntax

[*String* :=] *TNBXXmlClient.***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* =] *TNBXXmlClient.***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*

As Boolean = True], [Eab As Boolean = True])

Delphi Syntax

[*String* :=] *TNBXXmlClient.*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.

4.1.4.4.3.5 IsConnected

Returns a boolean value indicating if the emulation is on process.

Visual Basic Syntax

[*Boolean* =] *TNBXXmlClient.***IsConnected**

Delphi Syntax

[*Boolean* :=] *TNBXXmlClient.***IsConnected**;

See also **Connect** and **Disconnect** methods.

4.1.4.4.3.6 FindEditField

Gets the specified unprotected field, searching it by its name.

Visual Basic Syntax

[*TTnbField* =] *TNBXXmlClient.*FindEditField (caption As String)

Delphi Syntax

[*TTnbField* :=] *TNBXXmlClient.*FindEditField (caption:String);

Remarks

Caption parameter specifies the name of the label asociated 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:

String Value	Meaning
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

4.1.4.4.3.10 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

*TNBXXmlClient.***SendKeys (***Keys As String***)**

Delphi Syntax

*TNBXXmlClient.***SendKeys (***Keys:string***)**;

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

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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.

4.1.4.4.3.11 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.

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 is 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:

Constant Value	Meaning
akNone	No Action

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

*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.

4.1.4.5.2.3 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 colections.

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 maximun 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

*TNBXmlVirtual.***StartFileName** [= StartFileName As String]

Delphi Syntax

*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.

4.1.4.5.2.7 Text

Returns the entire screen.

VB Syntax

[*String* =] *TNBXmlVirtual.***Text**

Delphi Syntax

[*String* =] *TNBXmlVirtual.***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

*TNBXmlVirtual.***Connect**

Delphi Syntax

*TNBXmlVirtual.***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

*TNBXmlVirtual.***Disconnect**

Delphi Syntax

*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

[*Boolean* =] *TNBXmlVirtual.***IsConnected**

Delphi Syntax

[*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

[*Boolean* =] *TNBXmlVirtual.***IsLocked**

Delphi Syntax

[*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

[*Boolean* =] *TNBXmlVirtual.*LoadScreen (*xmlFileName As string*)

Delphi Syntax

[*boolean* :=] *TNBXmlVirtual.*LoadScreen (*xmlFileName:string*);

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

[*Boolean* =] *TNBXmlVirtual.*Press (*Value As string*)

Delphi Syntax

[*boolean* :=] *TNBXmlVirtual.*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:

String Value	Meaning
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.

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.

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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.

4.1.4.5.4 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:

```
TNBXTrace1.TelnetControl = Telnet
```

- Set the TCP/IP listening port to the **Port** property (at design time or at run time):

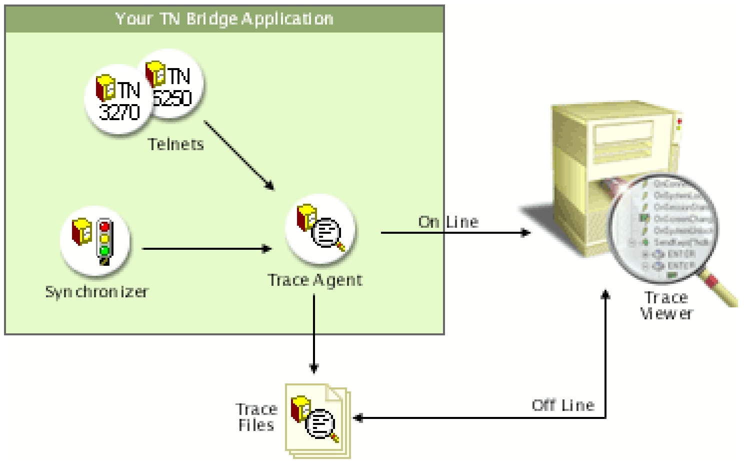
```
TNBXTrace1.Port = 1024
```

- Set the **Active** property to true. When doing this at design-time, the activation takes place at run-time:

```
TNBXTrace1.Active = True
```

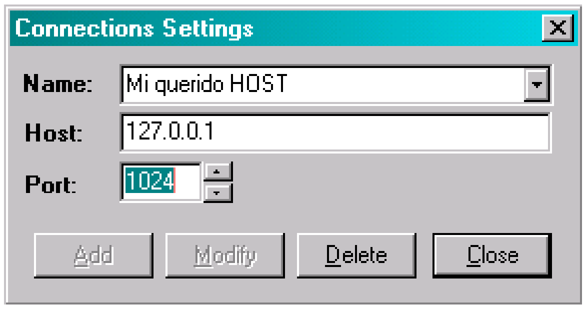
- Run your application.

- 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:

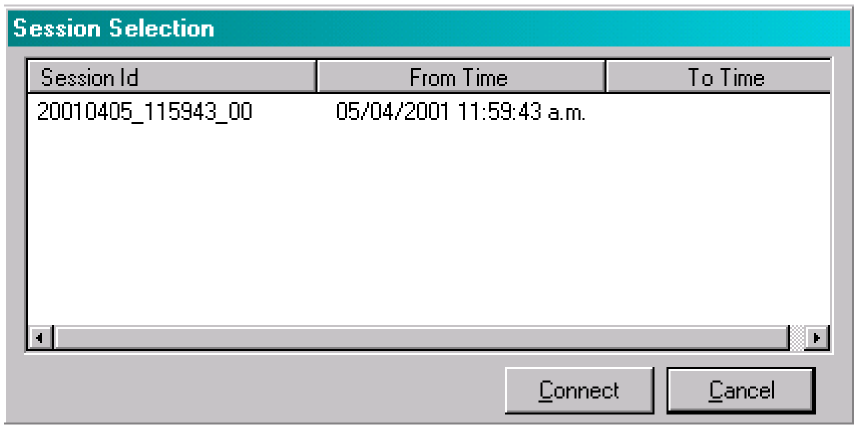

Connections Settings

Name: Mi querido HOST

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.



Session Id	From Time	To Time
20010405_115943_00	05/04/2001 11:59:43 a.m.	

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, 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

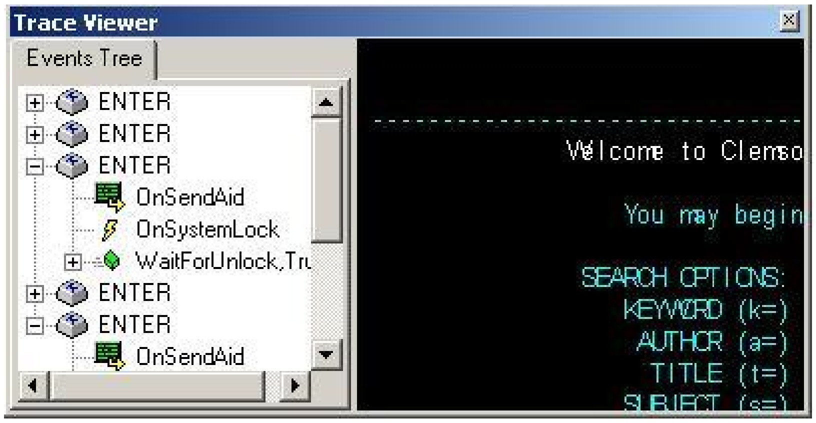
*TNBXTrace.***TelnetControl**; [:= *TNBnnnn*]

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.



```
-------------------------------
Welcome to Clemso
You may begin
SEARCH OPTIONS:
KEYWORD (k=)
AUTHOR (a=)
TITLE (t=)
SUBJECT (s=)
```

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

[*Variant* =] *TNBXTrace.***AsVariant**

Delphi Syntax

[*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

*TNBXTrace.***HideTrace**

Delphi Syntax

*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

*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 an its meaning.

Long Value	Meaning
0	Error
1	Info
2	StartPage
3	EndPage
4	StartMethod
5	EndMethod
6	SetProperty
7	SetVariable
8	Event
9	FieldsReceived
10	SendingFields
11	Connected
12	Disconnected
13	SendAid

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 Componentes
- 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.

[*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.

4.2.2 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:

Constant Value	Meaning
akNone	No Action
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 TNBXAid constants.

4.2.2.1.2.2 CodePage

Specifies the internal ASCII-EBCDIC conversion table.

Visual Basic Syntax

*TNBnnnn.***CodePage** [= *Integer*]

Delphi Syntax

*TNBnnnn.***CodePage** [:= *Integer*];

Remarks

It can take any of the Code Pages constant values:

Constant Value	Meaning
cpAustralia = 037	Australian
cpBelgium = 037	Belgium
cpCanada = 037	Canada

cpCzechRepublic = 870	Czech Republic
cpDenmark = 277	Denmark
cpFinland = 278	Finland
cpFrance = 297	France
cpGermany = 273	Germany
cpGreece = 423	Greece
cpGreece2 = 875	Greece
cpItaly = 280	Italy
cpLatinAmerica = 284	Latin America
cpNetherlands = 037	Netherlands
cpNorway = 277	Norway
cpPoland = 870	Poland
cpPortugal = 037	Portugal
cpSpain = 284	Spain
spSweden = 278	Sweden
cpSwitzerlandFrench = 500	Switzerland (French)
cpSwitzerlandGerman = 500	Switzerland (German)
cpUnitedKingdom = 285	United Kingdom
cpUnitedStates = 037	United States
cpInternational = 500	International

Default value is **cpUnitedStates.**

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

4.2.2.1.2.3 Cols

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

Visual Basic Syntax

[*Integer* =] TNBnnnn.**Cols**

Delphi Syntax

[*Integer* :=] 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:

```
[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.

4.2.2.1.2.6 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:

Constant Value	Meaning
True	Enables debug information.
False	Disables debug information.

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

*TNBnnnn.***Extended** [= *Boolean*]

Delphi Syntax

*TNBnnnn.***Extended** [:= *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 **Host**, **Port** and **TerminalType** properties.

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.

4.2.2.1.2.13 HostFields

Gets an array containing the screen fields.

Visual Basic Syntax

[**TNBField** =] *TNBnnnn.***HostFields (***Index As Variant***)**

Delphi Syntax

[**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

[*Boolean* =] *TNBnnnn.***IsExtended**

Delphi Syntax

[*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

*TNBnnnn.***KeepAlive** [= *Boolean*]

Delphi Syntax

*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

*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

[*Integer* =] *TNBnnnn*.**Rows**

Delphi Syntax

[*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

[*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

[**TNBSS** =] *TNBnnnn.***SessionState**

Delphi Syntax

[**TNBSS** :=] *TNBnnnn.***SessionState**;

It can return any of TNBSessionState constant values:

Constant Value	Meaning
ssNoSession	Not in session.
ssSSCPLU	In session with VTAM.
ssLULU	In session with VTAM Application.

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** [= **TNB5250TT**]

Delphi Syntax

*TNBnnnn.***TerminalType** [:= **TNB5250TT**];

It can take any of the following constant values:

For 5250:

Constant Value	Meaning
tt5211m2 = 0	IBM-5211-2 24 rows x 80 columns
tt3179m2 = 1	IBM-3179-2 24 rows x 80 columns
tt3477mFC = 2	IBM-3477-FC 27 rows x 132 columns
tt3180m2 = 3	IBM-3180-2 27 rows x 132 columns

Visual Basic Syntax

*TNBnnnnX.***TerminalType** [= TNB3270TT]

Delphi Syntax

*TNBnnnnX.***TerminalType**; [:= TNB3270TT]

For 3270:

Constant Value	Meaning
tt3278m2 = 0	IBM-3278-2 24 rows x 80 columns
tt3278m2E = 1	IBM-3278-2-E 24 rows x 80 columns extended
tt3278m3 = 2	IBM-3278-3 32 rows x 80 columns
tt3278m3E = 3	IBM-3278-3-E 32 rows x 80 columns extended
tt3278m4 = 4	IBM-3278-4 43 rows x 80 columns
tt3278m4E = 5	IBM-3278-4-E 43 rows x 80 columns extended

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

TNBnnnn.**XLockDelay** [:= *Integer*];

4.2.2.1.3 Methods

4.2.2.1.3.1 About

Shows Integration Pack about dialog box.

Visual Basic Syntax

*TNBnnnn.***About**

Delphi Syntax

*TNBnnnn.***About**;

4.2.2.1.3.2 ClassNameIs

Indicates if a specified name is the corresponding class name.

Visual Basic Syntax

[*Boolean* =] *TNBnnnn.***ClassNameIs (***Name As String***)**

Delphi Syntax

[*Boolean* :=] *TNBnnnn.***ClassNameIs (***Name: string***)**;

4.2.2.1.3.3 CloseEditor

Closes the modal dialog with the connection properties.

Visual Basic Syntax

*TNBEmulator.***CloseEditor**

Delphi Syntax

*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

*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

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

Delphi Syntax

[*String* :=] *TNBnnnn.***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.

4.2.2.1.3.8 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

[*String* :=] *TNBnnnn.***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.

4.2.2.1.3.9 IsConnected

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

Visual Basic Syntax

[*Boolean* =] *TNBnnnn.***IsConnected**

Delphi Syntax

[*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

*TNBnnnn.***LoadFromXMLFile** ***(****XMLFile as String***)**

Delphi Syntax

*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

*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:

String Value	Meaning
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.

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

*TNBnnnn.***SendKeys (***Keys As String***)**

Delphi Syntax

*TNBnnnn.***SendKeys (***Keys:string***)**;

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

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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.

See also **OnSystemUnlock** event.

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**

- OnEndDocument
- OnDataAvailable
- OnStartDocument

4.2.2.2.1.1 Properties

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

Visual Basic Syntax

*TTnb3287.***AttachMode** [= *TnbAttachMode*]

Delphi Syntax

*TTnb3287.***AttachMode** [:= *TnbAttachMode*];

Remarks

The following are possible values for AttachMode property:

Constant Value	Meaning
amDirect	Enables direct association between sessions.
amAssociate	The host handle the association between sessions.

Default value is **amDirect.**

Specifies the internal ASCII-EBCDIC conversion table.

Visual Basic Syntax

*TTnb3287.***CodePage** [= *TNBXCP*]

Delphi Syntax

*TTnb3287.***CodePage** [:= *TNBXCP*];

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

Constant Value	Meaning
cpAustralia = 037	Australian
cpBelgium = 037	Belgium
cpCanada = 037	Canada

cpCzechRepublic = 870	Czech Republic
cpDenmark = 277	Denmark
cpFinland = 278	Finland
cpFrance = 297	France
cpGermany = 273	Germany
cpGreece = 423	Greece
cpGreece2 = 875	Greece
cpItaly = 280	Italy
cpLatinAmerica = 284	Latin America
cpNetherlands = 037	Netherlands
cpNorway = 277	Norway
cpPoland = 870	Poland
cpPortugal = 037	Portugal
cpSpain = 284	Spain
spSweden = 278	Sweden
cpSwitzerlandFrench = 500	Switzerland (French)
cpSwitzerlandGerman = 500	Switzerland (German)
cpUnitedKingdom = 285	United Kingdom
cpUnitedStates = 037	United States
cpInternational = 500	International

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

TTnb3287.**ConversionTable** [= *String*]

Delphi Syntax

TTnb3287.**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

*TTnb3287.***Debug** [= *Boolean*]

Delphi Syntax

*TTnb3287.***Debug** [:= *Boolean*];

Remarks

The following are possible values for Debug property:

Constant Value	Meaning
True	Enables debug information.
False	Disables debug information.

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

*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

[*Boolean* =] *Tnb3287.***IsConnected**

Delphi Syntax

[*Boolean* :=] *Tnb3287.***IsConnected**;

See also **Connect** and **Disconnect** methods.

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

Visual Basic Syntax

*TTnb3287.***Port** [= *Integer*]

Delphi Syntax

*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

[*TTnbPrinter* =] *TTnb3287*.**Printer**

Delphi Syntax

[*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**;

4.2.2.2.1.3 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**
- **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:

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

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.

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

*TTnb3812.***Debug** [= *Boolean*]

Delphi Syntax

*TTnb3812.***Debug** [:= *Boolean*];

Remarks

The following are possible values for Debug property:

Constant Value	Meaning
True	Enables debug information.
False	Disables debug information.

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

[*Boolean* :=] *Tnb3812.***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

*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

[*String* =] *TTnb3812.***Text**

Delphi Syntax

[*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

*TTnb3812.***TnbProfiles** [= *TTnbProfiles*]

Delphi Syntax

*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

*TTnb3812.***Connect**

Delphi Syntax

*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

[*Boolean* =] *TNBField.***AutoEnter**

Delphi Syntax

[*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

[*Boolean* :=] *TNBField.***Blinking**

Delphi Syntax

[*Boolean* =] *TNBField.***Blinking**;

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

[*Boolean* =] *TNBField.***ByPass**

Delphi Syntax

[*Boolean* :=] *TNBField.***ByPass**;

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

*TNBField.***Col** [*Integer* =]

Delphi Syntax

*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

[*Integer* =] *TNBField.***Color**

Delphi Syntax

[*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

*TNBField.***Data** [= *String*]

Delphi Syntax

*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

[*Byte* =] *TNBField.***Eab**

Delphi Syntax

[*Byte* :=] *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

*TNBField.***Modified** [= *Boolean*]

Delphi Syntax

*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

[*Boolean* =] *TNBField.***MonoCase**

Delphi Syntax

[*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

[*Boolean* :=] *TNBField.***Numeric**;

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

*TNBField.*__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* =] *TNBField.*__Skip__

Delphi Syntax

[*Boolean* :=] *TNBField.*__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* =] *TNBField.*__Underline__

Delphi Syntax

[*Boolean* :=] *TNBField.*__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

[*Boolean* :=] *TNBField.***Visible**;

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* =] *TNBField*.**HllApiEab**

Delphi Syntax

[*Byte* :=] *TNBField*.**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 TTnbFields Class Reference

4.2.2.5.1 TTnbFields 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

[*Integer* =] *TNBFields*.**Count**

Delphi Syntax

[*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

[*TNBField* =] *TNBFields*.**CursorField**

Delphi Syntax

[*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

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**

Methods

- **Abort**
- **LoadFromXMLFile**
- **Receive**
- **Run**
- **SaveToXMLFile**
- **Send**

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

TNBIndFile.**Append**; [:= Boolean]

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 (**Lrecl** 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, **Lrecl** 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

TNBIndFile.**BufSize** [= Integer]

Delphi Syntax

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

TNBIndFile.**Bytes** [Integer =]

Delphi Syntax

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

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:

Constant Value	Meaning
ftDFT	DFT mode
ftCut	Cut mode

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:

Constant Value	Meaning
hVM	VM
hTSO	TSO
hCics	Cics

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:

Value	Meaning
0	Default file format
1	Fixed file format
2	Variable file format
3	Undefined file format

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

TNBIndFile.**SecSpace** [= Integer]

Delphi Syntax

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

TNBIndFile.**ShowDialog** [= Boolean]

Delphi Syntax

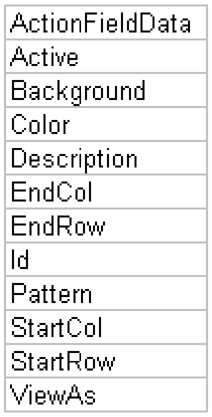
TNBIndFile.**ShowDialog**; [:= Boolean]

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.



ActionFieldData
Active
Background
Color
Description
EndCol
EndRow
Id
Pattern
StartCol
StartRow
ViewAs

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).

Value	Meaning
0	Default unit of allocation
1	Tracks unit of allocation
2	Cylinder unit of allocation
3	Average Blocks unit of allocation

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

TNBIndFile.**Abort**

Delphi Syntax

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

*TNBIndFile.***LoadFromXMLFile (***XMLFile As String***)**

Delphi Syntax

*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

TNBIndFile.**Receive**

Delphi Syntax

TNBIndFile.**Receive***;*

See also **Direction** property, **Run** method, **Send** method.

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 *TNBIndFile* object that raises the event.

See also **OnAborting** event.

4.2.2.6.4.3 OnRunning

Occurs when the file transfer operation has been started.

Declaration

procedure OnRunning (Sender: TObject)

Parameters

Sender: The *TNBIndFile* object that raises the event.

See also **OnComplete** event.

4.2.2.6.4.4 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 *TNBIndFile* object that raises the event.

See also **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.

Constant Value	**Meaning**
ftDFT	DFT mode
ftCut	Cut mode

4.2.2.6.5.2 THostKind

These values are used in the **HostKind** property of **TTnbIndFile** control.

Constant Value	**Meaning**
hVM	VM
hTSO	TSO
hCics	Cics

4.2.2.7 Constants

4.2.2.7.1 TNBCP

These values are used in the **CodePage** property of telnet components.

Constant Value	**Meaning**
cpAustralia = 037	Australian
cpBelgium = 037	Belgium
cpCanada = 037	Canada
cpCzechRepublic = 870	Czech Republic
cpDenmark = 277	Denmark
cpFinland = 278	Finland
cpFrance = 297	France
cpGermany = 273	Germany
cpGreece = 423	Greece
cpGreece2 = 875	Greece
cpItaly = 280	Italy
cpLatinAmerica = 284	Latin America
cpNetherlands = 037	Netherlands
cpNorway = 277	Norway
cpPoland = 870	Poland
cpPortugal = 037	Portugal

cpSpain = 284	Spain
spSweden = 278	Sweden
cpSwitzerlandFrench = 500	Switzerland (French)
cpSwitzerlandGerman = 500	Switzerland (German)
cpUnitedKingdom = 285	United Kingdom
cpUnitedStates = 037	United States
cpInternational = 500	International

4.2.2.7.2 TNBAid

These values are used in the **AidKey** property of telnet components.

Constant Value	Meaning
akNone	No Action
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

4.2.2.7.3 TNBSS

These values are used in the **SessionState** property of telnet components.

Constant Value	Meaning
ssNoSession	Not in session.
ssSSCPLU	In session with VTAM.
ssLULU	In session with VTAM Application.

4.2.2.7.4 TNB5250TT

These values are used in the **TerminalType** property of TNB5250 component.

Constant Value	Meaning
tt5211m2 = 0	IBM-5211-2 24 rows x 80 columns
tt3179m2 = 1	IBM-3179-2 24 rows x 80 columns
tt3477mFC = 2	IBM-3477-FC 27 rows x 132 columns
tt3180m2 = 3	IBM-3180-2 27 rows x 132 columns

4.2.2.7.5 TNB3270TT

These values are used in the **TerminalType** property of TNB3270 component.

Constant Value	Meaning
tt3278m2 = 0	IBM-3278-2 24 rows x 80 columns
tt3278m2E = 1	IBM-3278-2-E 24 rows x 80 columns extended
tt3278m3 = 2	IBM-3278-3 32 rows x 80 columns
tt3278m3E = 3	IBM-3278-3-E 32 rows x 80 columns extended
tt3278m4 = 4	IBM-3278-4 43 rows x 80 columns
tt3278m4E = 5	IBM-3278-4-E 43 rows x 80 columns extended

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

*TNBEmulator.***DisableKbdLockOnError** [= *Boolean*]

Delphi Syntax

*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

*TNBEmulator.***EnableSelection** [= *Boolean*]

Delphi Syntax

*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 wherethe 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

*TNBEmulator.***InputInhibitEnabled** [= *Boolean*]

Delphi Syntax

*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

*TNBEmulator.***InsertMode** [= *Boolean*]

Delphi Syntax

*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.

Visual Basic Syntax

[*Boolean* =] *TNBEmulator.***IsInputInhibited**

Delphi Syntax

[*Boolean* :=] *TNBEmulator.***IsInputInhibited**;

See Also **InputInhibitEnabled** property and **OnInputInhibitChange** event.

4.2.3.1.2.10 LeftMargin

Sets emulation screen left margin.

Visual Basic Syntax

*TNBEmulator.***LeftMargin** [= *Integer*]

Delphi Syntax

*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

*TNBEmulator.***RulerType** [= **TTnbRulerType**]

Delphi Syntax

*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:

Constant Value	Meaning
rtNone = 0	No lines
rtCrosshair = 1	Horizontal and vertical lines
rtVertical = 2	Only vertical line
rtHorizontal = 3	Only horizontal line

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

*TNBEmulator.***StyleName** [= *String*]

Delphi Syntax

*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

*TNBEmulator.***SubKey** [= *String*]

Delphi Syntax

*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

*TNBEmulator.***TnbCom** [*:= TNBnnnn*];

Remarks

You must set this property for TNBEmulator control to allow it to work properly.

4.2.3.1.2.16 TnbHotSpots

Sets the **TNBHotSpot** component as the hotspots handler for this control.

Visual Basic Syntax

*TNBEmulator.***TnbHotSpots** [= **TNBHotSpot**]

Delphi Syntax

*TNBEmulator.***TnbHotSpots** [*:=* **TNBHotSpot**];

See Also **TNBHotSpot** component.

4.2.3.1.2.17 TnbProfiles

Sets the component as the profile manager for this control.

Visual Basic Syntax

*TNBEmulator.***TnbProfiles** [= **TNBProfiles**]

Delphi Syntax

*TNBEmulator.***TnbProfiles** [*:=* **TNBProfiles**];

Remarks

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

See Also **TNBProfiles** component, **StyleName** property and **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

*TNBEmulator.***About**

Delphi Syntax

*TNBEmulator.***About**;

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.

4.2.3.1.3.3 AsVariant

Returns the control as a variant data type variable.

Visual Basic Syntax

[*Variant* =] *TNBEmulator.***AsVariant**

Delphi Syntax

[*Variant* :=] *TNBEmulator.***AsVariant**;

4.2.3.1.3.4 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

[*String* :=] *TNBEmulator.*__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.

4.2.3.1.3.8 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.

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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 **OnDblClick** 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.

Constant Value	**Meaning**
rtNone = 0	No lines
rtCrosshair = 1	Horizontal and vertical lines
rtVertical = 2	Only vertical line
rtHorizontal = 3	Only horizontal line

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

Default value is **False**.

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

*TNBAidHook.***KbdKind** [= **TNBKbdKind**]

Delphi Syntax

*TNBAidHook.***KbdKind** [*:*= **TNBKbdKind**];

It can take one of this constants values:

Constant Value	Meaning
k5250	5250 Keyboard
k3270	3270 Keyboard

Remarks

This component maintain separates 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

*TNBAidHook.***Mapping3270** [= OleVariant]

Delphi Syntax

*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.

4.2.3.2.2.11 TnbProfiles

Sets the **TNBProfiles** Component as the profile manager for this control.

Visual Basic Syntax

*TNBAidHook.***TnbProfiles** [= ***TNBProfiles***]

Delphi Syntax

*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

[*Boolean* =] *TNBAidHook.***ClassNameIs (***Name As String***)**

Delphi Syntax

[*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:

Value	Meaning
Up	go to previous row
Down	go to next row
Left	go to previous column
Right	go to next column
Insert	set the keyboard in inserting state
Delete	delete the next character
BackSpace	delete the previous character
ScreenBegin	go to row 1 column 1
ScreenEnd	go to last row last column
NextLine	go to column 1 of the next line
PriorField	go to the previous field
NextField	go to the next field
Home	go to the first field
End	go to last character of current field
EraseField	erase the current field
EraseEof	erase characters from current position until the end of field
FieldMinus	skip to the previous field
FieldPlus	skip to the next field
Restore	restore the current screen

Host processing function keys (Attention Identifier "AID" keys):

Value	Meaning

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.

Constant Value	Meaning
k5250	5250 Keyboard
k3270	3270 Keyboard

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

*TNBStatusBar.***Font** [= *IFontDisp*]

Delphi Syntax

*TNBStatusBar.***Font** [*:=* *IFontDisp*];

4.2.3.3.2.2 TnbCom

Sets the TNB3270 or TNB5250 component as the telnet client component.

Visual Basic Syntax

*TNBStatusBar.***TnbCom** [= *TNBnnnn*]

Delphi Syntax

*TNBStatusBar.***TnbCom** [*:=* *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

[**TNBMacroState** =] *TNBMacro*.**State**

Delphi Syntax

[**TNBMacroState** :=] *TNBMacro*.**State**;

Remarks

The value of this property can be one of the following constants:

Constant Value	**Meaning**
msNoMacro	No macro is loaded.
msStopped	The component has a macro loaded and is in a stopped state.
msRecording	The component is recording a macro.
msPlaying	The component is playing the macro loaded.

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.***SubKey** [= *String*]

Delphi Syntax

*TNBMacro.***SubKey** [:= *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

*TNBMacro.***StartPlaying (***Name as String***)**

Delphi Syntax

*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

*TNBMacro.***StartRecording()**

Delphi Syntax

*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 same cases the macro doesn't stop automatically because of changes on the screen sequences or field positions.

See also **StartPlaying** method **OnStopPlaying** event.

4.2.3.4.3.7 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.

Constant Value	**Meaning**
msNoMacro	No macro is loaded.
msStopped	The component has a macro loaded and

	is in a stopped state.
msRecording	The component is recording a macro.
msPlaying	The control is playing the macro loaded.

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

[*OleVariant* =]*TNBProfiles.***Items (***SubKey As String***)**

Delphi Syntax

[*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

*TNBProfiles.***Delete (***SubKey As String, Profile As String***)**

Delphi Syntax

*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

[*String* :=] *TNBProfiles*.**GetCurrent *(****Source As String****)***

Delphi Syntax

[*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

TNBProfiles.**GetXMLHeader (***Tag As String, Name As String, Id As String, Type_ As*

*String***)**

TNBProfiles.**GetXMLHeader** ***(****Filename As String, Tag As String, Name As String, Id As String, Type_ As String****)***

Delphi Syntax

TNBProfiles.**GetXMLHeader** ***(****var Tag, Name, Id, Type_: string***);**

TNBProfiles.**GetXMLHeader** ***(****Filename: string; var Tag, Name, Id, Type_: string***);**

4.2.3.5.3.6 Rename

Renames the specified profile.

Visual Basic Syntax

*TNBProfiles.***Rename** ***(****Source As String, OldProfile As String, Profile As String, Id As String= ' ', type_ As String = ' '****)***

Delphi Syntax

*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

*TNBProfiles.***Rollback**

Delphi Syntax

*TNBProfiles.***Rollback**;

See also **Commit** method.

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**

- **EndCol**
- **EndRow**
- **Id**
- **Pattern**
- **StartCol**
- **StartRow**
- **TextColor**
- **ViewAs**

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

*TNBHotSpot.***Active** [= *Boolean*]

Delphi Syntax

*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

*TNBHotSpot.***Background** [= *OLE_COLOR*]

Delphi Syntax

*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

*TNBHotSpot.***Color** [= *TColor*]

Delphi Syntax

*TNBHotSpot.***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

*TNBHotSpot.***Description** [= *String*]

Delphi Syntax

*TNBHotSpot.***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

*TNBHotSpot.***StartCol** [= *Integer*]

Delphi Syntax

*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

*TNBHotSpot.***StartRow** [= *Integer*]

Delphi Syntax

*TNBHotSpot.***StartRow** [:= *Integer*];

Remarks

Valid values are from 1 to *TNB5250/TNB3270.***Rows** property.

See Also **EndRow**, **EndCol**, **StartCol** properties.

4.2.3.6.2.12 TextColor

Sets the hotspot foreground color to be shown in a host screen.

Visual Basic Syntax

*TNBHotSpot.***TextColor** [= *OLE_COLOR*]

Delphi Syntax

*TNBHotSpot.***TextColor** [:= *OLE_COLOR*];

Remarks

This property is a long value that represent a hotspot foreground color.

See Also **Background** and **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.

Visual Basic Syntax

*TNBHotSpot.***ViewAs** [= **TTnbHotSpotViewAs**]

Delphi Syntax

*TNBHotSpot.***ViewAs** [:= **TTnbHotSpotViewAs**];

It can take one of this constants values:

Constant Value	Meaning
hsButton	Hotspot button style is displayed.
hsLink	Hotspot link style is displayed.

hsNone	No hotspot is displayed.
hsPlain	Hotspot plain text style is displayed.

4.2.3.6.3 Constants

4.2.3.6.3.1 TTnbHotSpotViewAs

These values are used in the **ViewAs** property of hotspot component.

Constant Value	**Meaning**
hsButton	Hotspot button style is displayed.
hsLink	Hotspot link style is displayed.
hsNone	No hotspot is displayed.
hsPlain	Hotspot plain text style is displayed.

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.***Cols** 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

[*Integer* =] *TNBHotSpots.***StartCol**

Delphi Syntax

[*Integer* :=] *TNBHotSpots.***StartCol**;

Remarks

Return value is between 1 and *TNB5250/TNB3270.***Cols** property.

See Also **EndRow**, **EndCol**, **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

[*Integer* =] *TNBHotSpots.***StartRow**

Delphi Syntax

[*Integer* :=] *TNBHotSpots.***StartRow**;

Remarks

Return value is between 1 and *TNB5250/TNB3270.***Rows** property.

See Also **EndCol**, **StartCol**, **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 **TnbProfiles** property has been assigned to a **TNBProfiles** component.

See also **TnbProfiles** property and **TNBProfiles** component.

4.2.3.7.2.8 TnbProfiles

Sets the **TNBProfiles** component as the profile manager for this control.

Visual Basic Syntax

*TNBHotSpots.***TnbProfiles** [= **TNBProfiles**]

Delphi Syntax

*TNBHotSpots.***TnbProfiles** [:= **TNBProfiles**];

See Also **TNBProfiles** component and **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**

Delphi Syntax

[**TTnbHotSpotViewAs** :=] *TNBHotSpot.***ViewAs**;

It can take one of this constant values:

Constant Value	Meaning
hsButton	Hotspot button style is displayed.
hsLink	Hotspot link style is displayed.
hsNone	No hotspot is displayed.
hsPlain	Hotspot plain text style is displayed.

4.2.3.7.3 Methods

4.2.3.7.3.1 About

Shows Integration Pack about dialog box.

Visual Basic Syntax

*TNBHotSpots.***About**

Delphi Syntax

*TNBHotSpots.***About**;

4.2.3.7.3.2 Add

Adds a new TnbXHotspot to the **TNBHotSpots** collection.

Visual Basic Syntax

*[TnbHotSpot =] TNBXHotSpots.***Add**

Delphi Syntax

*[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

*TNBSync.***EmptyScreenDelay** [= *Integer*]

Delphi Syntax

*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

*TNBSync.***TnbCom** [= *TNBnnnn*]

Delphi Syntax

*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

*[Boolean] := TNBSync.***Connect(***[TimeOut] As Integer***)**

Delphi Syntax

*[Boolean] := 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

*[Boolean] := TNBSync.***Disconnect(***[TimeOut] As Integer***)**

Delphi Syntax

*[Boolean] := 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.

Code Value	**Meaning**
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

This method complements the **SendKeys** method from **TNB3270** components 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

*[Boolean] := TNBSync.***WaitForDisconnect (***[TimeOut] As Integer***)**

Delphi Syntax

*[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

*[Boolean] = TNBSync.***WaitForNewScreen (***[TimeOut] As Integer***)**

Delphi Syntax

*[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

*[Boolean] = TNBSync.***WaitForNewUnlock (***[TimeOut] As Integer***)**

Delphi Syntax

*[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

*[Boolean] = TNBSync.***WaitForStartDoc (***[TimeOut] As Integer***)**

Delphi Syntax

*[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

*[Boolean] := TNBSync.***WaitForScreen (***[TimeOut] As Integer***)**

Delphi Syntax

*[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

*[Boolean] := TNBSync.***WaitForUnlock(***[TimeOut] As Integer***)**

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

- OnCustomXml

4.2.4.2.2 Properties

4.2.4.2.2.1 Id

Gets an Id for the active XML Template.

VB.NET Syntax

[*String* =] *XmlTemplate.***Id**

C# Syntax

[*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

[*Boolean* =] *XmlTemplate.***CanProduce**

C# Syntax

[*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

[*String* =] *XmlTemplate.***HostToXml**

C# Syntax

[*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

*XmlTemplate.***HostToXml**(Xml As String)

C# Syntax

*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 TNBXmlBroker 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

TNBXmlBroker.**IncludeStatus** [= *Boolean*]

Delphi Syntax

*TNBXmlBroker.***IncludeStatus** [:= *boolean*];

4.2.4.3.2.2 Producer

Assigns an XML object capable to produce XML content.

Visual Basic Syntax

*TNBXmlBroker.***Producer**[= *IXmlProducer*]

Delphi Syntax

*TNBXmlBroker.***Producer** [:= *IXmlProducer*];

See also **TTnbXmlTemplate** class

4.2.4.3.2.3 ScreenName

Sets/gets the corresponding screen name.

Visual Basic Syntax

*TNBXmlBroker.***ScreenName** [= *String*]

Delphi Syntax

*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

*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:

```
<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 />
```

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 propery 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:

```
<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">
                               <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>
```

4.2.4.4 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 **TNBXmlBroker** 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 comunication 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

*TNBXmlClient.***AIDKey** [:= **TNBAid**];

It can take any of TNBAidKey constant values:

Constant Value	**Meaning**
akNone	No Action
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 **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 necesary.

Visual Basic Syntax

*TNBXmlClient.***AIDString** [= AidKey As String]

Delphi Syntax

*TNBXmlClient.***AIDString** [:= AidKey:String];

The following table lists the possible string values:

String Value	Meaning
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 SendAid method.

4.2.4.4.2.3 ExcludeEmptyFields

Sets/gets the possibility of excluding all empty fields.

Visual Basic Syntax

*TNBXmlClient.***ExcludeEmptyFields** [= Boolean]

Delphi Syntax

*TNBXmlClient.***ExcludeEmptyFields** [*:*= Boolean];

4.2.4.4.2.4 ScreenRow

Gets the text contained into the specified row.

Visual Basic Syntax

[*String* =] *TNBXmlClient.***ScreenRow (***Index As Integer***)**

Delphi Syntax

[*String* :=] *TNBXmlClient.***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

[*String* =] *TNBXmlClient.***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

*TNBXmlClient.***Connect**

Delphi Syntax

*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

*TNBXmlClient.***Disconnect**

Delphi Syntax

*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

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

Delphi Syntax

[*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 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.

4.2.4.4.3.4 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

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

Delphi Syntax

[*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 asociated 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

*TNBXmlClient.***LoadFromXMLFile** ***(****XMLFile As String****)***

Delphi Syntax

*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

*TNBXmlClient.***SaveToXMLFile** ***(****XMLFile As String****)***

Delphi Syntax

*TNBXmlClient.***SaveToXMLFile** ***(****XMLFile: string****);***

4.2.4.4.3.9 SendAid

Sets an Attention Identifier Key to the **AidSring** property and fires the **OnSendAid** event.

Visual Basic Syntax

*TNBXmlClient.***SendAid (***AidKey As string***)**

Delphi Syntax

*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:

String Value	Meaning
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

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.

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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.

4.2.4.4.4.4 OnDisconnect

Occurs when the **SetConnected** property is set to *False*.

See also **Disconnect** and **SetConnected** methods, **OnConnect**, **OnConnectRemote**, **OnDisconnect** and **OnDisconnectRemote** event.

4.2.4.4.4.5 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.

4.2.4.4.4.6 OnScreenChange

Occurs when an valid XML is assigned, either thru **XML** is property or LoadFromXmlFile method.

See also **XML** property, **OnSystemLock** and **OnSystemUnlock** events.

4.2.4.4.4.7 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**

- BaseDirectory
- CurrentScreen
- ExcludeEmptyFields
- FieldSplitting
- StartFilename
- Text
- XML

Methods

- Connect
- Disconnect
- IsConnected
- IsLocked
- IsScreenEmpty
- GetText
- LoadScreen
- Press
- SetConnected
- SetText
- Type

Events

- OnConnect
- OnConnectionFail
- OnDisconnect
- OnNewScreen
- OnScreenChange
- OnSendAid
- OnSystemLock
- OnSystemUnlock

4.2.4.5.2 Properties

4.2.4.5.2.1 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:

Constant Value	Meaning
akNone	No Action
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.

4.2.4.5.2.2 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

*TNBXmlVirtual.***ExcludeEmptyFields** [:= ExcludeEmptyFields:Boolean];

4.2.4.5.2.5 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 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

*TNBXmlVirtual.***StartFileName** [= StartFileName As String]

Delphi Syntax

*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* =] *TNBXmlVirtual.***Text**

Delphi Syntax

[*String* =] *TNBXmlVirtual.***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

*TNBXmlVirtual.***XML** [= *String*]

Delphi Syntax

*TNBXmlVirtual.***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

*TNBXmlVirtual.***Connect**

Delphi Syntax

*TNBXmlVirtual.***Connect**;

See also **Connect**...

See also **Disconnect** method, **OnConnect** and **OnConnectionFail** events.

4.2.4.5.3.2 Disconnect

The **Disconnect** method ends the connection sequence process.

Visual Basic Syntax

*TNBXmlVirtual.***Disconnect**

Delphi Syntax

*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

[*Boolean* =] *TNBXmlVirtual.***IsConnected**

Delphi Syntax

[*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

[*Boolean* =] *TNBXmlVirtual.***IsLocked**

Delphi Syntax

[*boolean* :=] *TNBXmlVirtual.***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

[*Boolean* =] *TNBXmlVirtual.***IsScreenEmpty**

Delphi Syntax

[*boolean* :=] *TNBXmlVirtual.***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

[*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.

4.2.4.5.3.7 LoadScreen

Loads an XML file to provide with new screen content.

VB Syntax

[*Boolean* =] *TNBXmlVirtual.***LoadScreen (***xmlFileName As string***)**

Delphi Syntax

[*boolean* :=] *TNBXmlVirtual.***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* =] *TNBXmlVirtual.***Press (***Value As string***)**

Delphi Syntax

[*boolean* :=] *TNBXmlVirtual.***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:

String Value	Meaning
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.

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.

Code Value	Meaning
@A@Q	Attention
@<	Backspace
@B	BackTab (Left Tab)
@C	Clear
@E	Enter
@F	Erase Field
@A@Q	Sys Request
@T	Tab (Right Tab)
@x	PA1
@y	PA2
@z	PA3
@1	PF1
@2	PF2
@3	PF3
@4	PF4
@5	PF5
@6	PF6
@7	PF7
@8	PF8
@9	PF9
@a	PF10
@b	PF11
@c	PF12
@d	PF13
@e	PF14
@f	PF15
@g	PF16
@h	PF17
@i	PF18

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 analize **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 an 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):

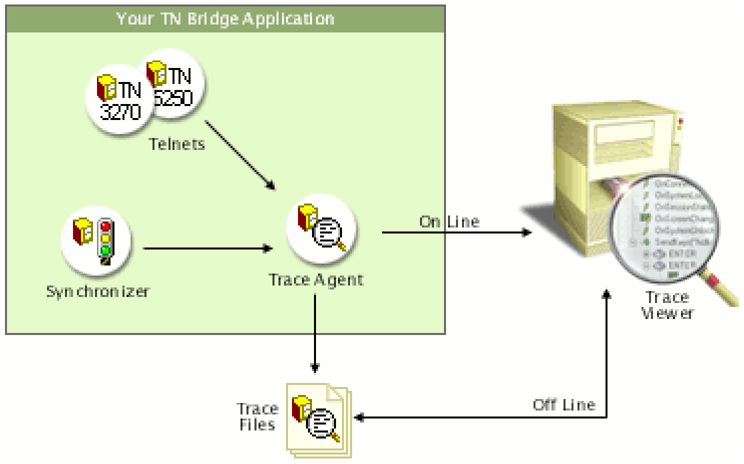
```
TNBTrace1.Port := 1024
```

- Set the **Active** property to true. When doing this at design-time, the activation takes place at run-time:

```
TNBTrace1.Active := True
```

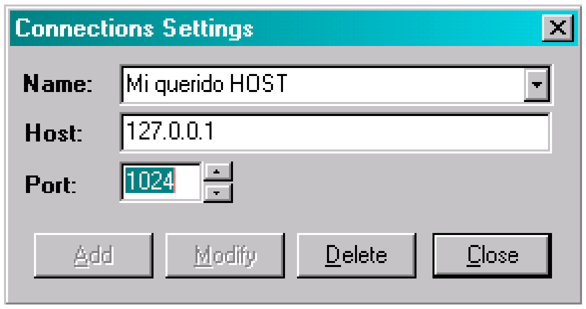
- Run your application.

- 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**.



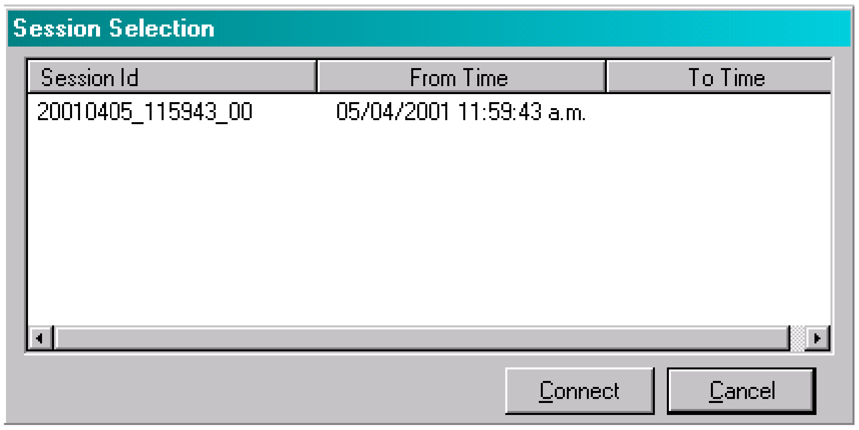
For on-line monitoring:

- 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:



Name: Mi querido HOST
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.



Session Id	From Time	To Time
20010405_115943_00	05/04/2001 11:59:43 a.m.	

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, is 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

*TNBTrace.***Directory** [= *String*]

Delphi Syntax

*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

*TNBTrace.***HidePasswordFields** [= *Boolean*]

Delphi Syntax

*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

*TNBTrace.***TnbCom** [= *TNBnnnn*]

Delphi Syntax

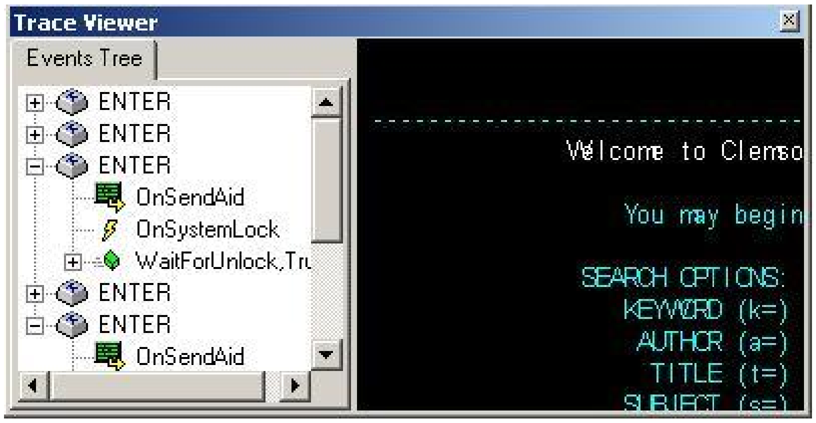
*TNBTrace.***TnbCom**; [:= *TNBnnnn*]

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.



```
------------------------------
        Welcome to Clemso

            You may begin

          SEARCH OPTIONS:
             KEYWORD (k=)
              AUTHOR (a=)
               TITLE (t=)
             SUBJECT (s=)
```

Visual Basic Syntax

*TNBTrace.***Visible** [= *Boolean*]

Delphi Syntax

*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.

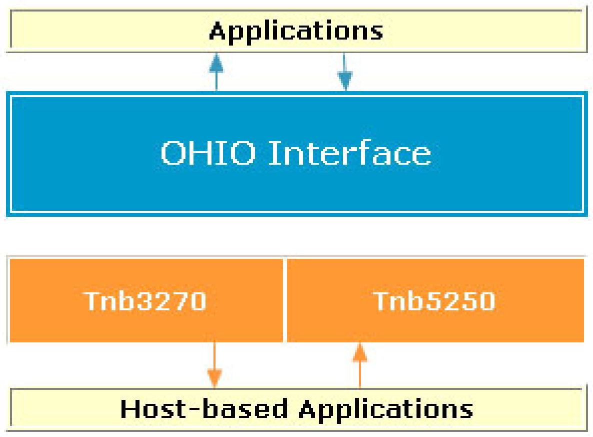
5 OHIO Reference

- **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:

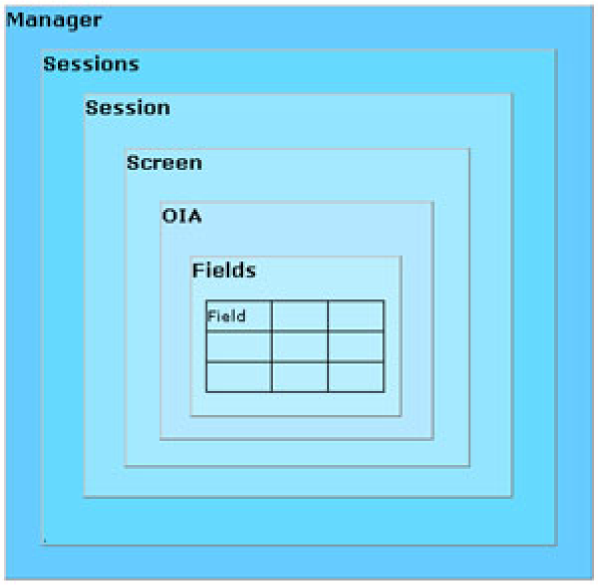
- You can run an OHIO application without having an emulation session running.
- OHIO has all the benefits of the object oriented programming paradigm because it was designed as an object oriented API.
- You can concentrate on the application functions, without worrying about structure packing details 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:



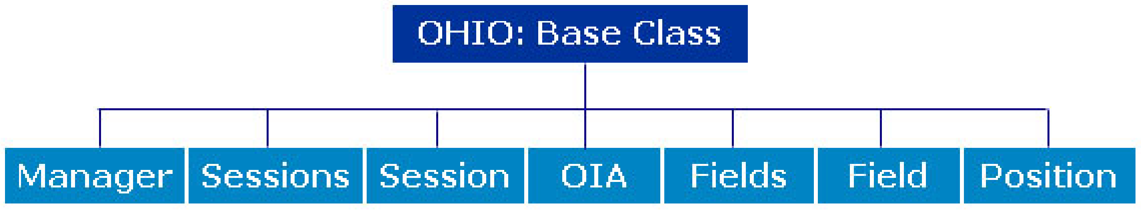
Additional utility classes:

- **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:



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

*[TTnbOhioPosition =] OhioObject.***CreateOhioPosition(**row As Integer, col As Integer**)**

Delphi Syntax

*[TTnbOhioPosition :=] OhioObject.***CreateOhioPosition(**row,col:integer**)**;

Parameters

Parameter	Description
row	The row coordinate
col	The column coordinate

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

*[String =] OhioObject.***OhioVersion**

Delphi Syntax

*[String :=] 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

[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

[TTnbTelnet =] Screen.**VendorObject**

[TTnbTelnet =] Session.**VendorObject**

[TTnbOIA =] OIA.**VendorObject**

[TTnbSessions =] Sessions.**VendorObject**

[TTnbField =] Field.**VendorObject**

Delphi Syntax

*[TTnbTelnet :=] Screen.***VendorObject**;

[TTnbTelnet :=] Session.**VendorObject**;

[TTnbOIA :=] OIA.**VendorObject**;

[TTnbSessions :=] Sessions.**VendorObject**;

*[TTnbField :=] 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

[String :=] OhioObject.**VendorProductVersion**;

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

*[TTnbOhioSession =] Manager.***OpenSession(**ConfigurationResource As Integer, SessionName As String**)**

*[TTnbOhioSession =] Manager.***OpenSession(**ConfigurationResource As TTnbComApi, SessionName As String**)**

*[TTnbOhioSession =] Manager.***OpenSession(**ConfigurationResource As String, SessionName As String**)**

Delphi Syntax

*[TTnbOhioSession :=] Manager.***OpenSession(**ConfigurationResource:integer; SessionName:string**)**;

*[TTnbOhioSession :=] Manager.***OpenSession(**ConfigurationResource:TTnbComApi; SessionName:string**)**;

*[TTnbOhioSession :=] Manager.***OpenSession(**ConfigurationResource, SessionName:string**)**;

Parameters

Parameter	**Description**
ConfigurationResource	A vendor specific string used to provide configuration information
SessionName	The unique name associated with an OhioSession

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 methods 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

Parameters	**Description**
SessionObject	The OhioSession to close.
SessionName	The SessionName of the OhioSession to close.

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

*[OhioSessions =] Manager.***Sessions**

Delphi Syntax

*[OhioSessions :=] Manager.***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

[TTnbOhioSession =] Sessions.**AddSession(**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, SessionName:string**)***;*

*[TTnbOhioSession :=] Sessions.***AddSession(**ConfigurationResource:integer; SessionName:string**)***;*

*[TTnbOhioSession :=] Sessions.***AddSession(**ConfigurationResource:TTnbComApi; SessionName:string**)***;*

Parameters

Parameters	**Description**
ConfigurationResource	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.
SessionName	Session's name.

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

Sessions.**CloseSession(**SessionObject As TTnbOhioSession**)**

Sessions.**CloseSession(**SessionName As String**)**

Delphi Syntax

Sessions.**CloseSession(**SessionObject:TTnbOhioSession**)**;

Sessions.**CloseSession(**SessionName:string**)**;

Parameters

Parameters	**Description**
SessionObject	The OhioSession to close.

SessionName	The SessionName of the OhioSession to close.

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

*[Integer =] Sessions.***Count**

Delphi Syntax

*[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

*[TTnbOhioSession] = Sessions.***Item(**index As Variant**)**

Delphi Syntax

*[TTnbOhioSession] := Sessions.***Item(**index:Variant**)**;

Parameters

Parameters	**Description**

index	The index of the target OhioSession.

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**

Delphi Syntax

*Session.***Connect**;

Parameters

None.

5.2.2.4.1.2 Disconnect

Stops the communications link to the host.

Visual Basic Syntax

Session.**Disconnect**

Delphi Syntax

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

*[Boolean =] Session.***WaitForConnect(**[TimeOut] As Integer**)**

Delphi Syntax

*[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

*[String :=] Session.***ConfigurationResource***;*

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

*Session.***Manager**; *[:= OhioManager]*

5.2.2.4.2.4 Screen

Returns the OhioScreen object for the current session.

Visual Basic Syntax

*[OhioScreen =] Session.***Screen**

Delphi Syntax

*[OhioScreen :=] Session.***Screen**;

5.2.2.4.2.5 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.

5.2.2.4.2.6 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 TNBXSync 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**

- **WaitForNewScreen**
- **WaitForNewUnlock**
- **WaitForScreen**
- **WaitForUnlock**

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

Parameter	**Description**
start	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). "start" must be positionally less than "end".
end	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). "end" must be positionally greater than "start".
plane	A valid OHIO_PLANE value.

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

Parameter	**Description**
TargetString	The target string.
startPos	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).
length	The length from startPos to include in the search.
dir	An OHIO_DIRECTION value.
IgnoreCase	Indicates whether the search is case sensitive. True means that case will be ignored. False means the search will be case sensitive.

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 though 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

Parameters	Description
aidKey	The aid key to send to the virtual screen.

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

Parameters	Description
text	The string of characters to be sent.
location	OhioPosition where the string should be written.

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

*Screen.***SetString(**text As String, location As TTnbOhioPosition**)**

Delphi Syntax

*Screen.***SetString(**text:string; location:TTnbOhioPosition**)**

Parameters

Parameters	**Description**
text	String to place in the virtual screen
location	Position where the string should be written

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

[Boolean] = *Screen.***Wait(**[TimeOut] As Integer**)**

Delphi Syntax

[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

*[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

*[WordBool :=] Screen.***WaitFor(**Value:string;[TimeOut]:Integer; ResetWait:Boolean**)**;

*[Integer :=] Screen.***WaitFor(**const Values:Array of string;[TimeOut]:Integer; ResetWait:Boolean**)**;

Parameters

Parameters	**Description**
Value	The string of characters expected to be found.
Values	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 ";"
TimeOut	Optional parameter measured in milliseconds.
ResetWait	While this parameter is false, continue lopping until the expected string is found.

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

*[Boolean =] Screen.***WaitForNewScreen(**[TimeOut] As Integer**)**

Delphi Syntax

*[WordBool :=] Screen.***WaitForNewScreen(**[TimeOut]: Integer**)**;

Parameters

Parameters	**Description**
TimeOut	Optional parameter measured in milliseconds.

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

*[WordBool :=] Screen.***WaitForNewUnlock(**[TimeOut]: Integer**)**;

Parameters

Parameters	**Description**
TimeOut	Optional parameter measured in milliseconds.

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

*[WordBool :=] Screen.***WaitForScreen(**[TimeOut]: Integer**)**;

Parameters

Parameters	**Description**
TimeOut	Optional parameter measured in milliseconds.

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

Parameters	**Description**
TimeOut	Optional parameter measured in milliseconds.

5.2.2.5.2 Properties

- **Columns**
- **Cursor**
- **Fields**

- **OIA**
- **Rows**
- **String/Text**

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

*[TTnbOhioFields =] Screen.***Fields**

Delphi Syntax

*[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

*[TTnbOhioOIA =] Screen.***OIA**

Delphi Syntax

*[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

*[Integer =] Screen.***Rows**

Delphi Syntax

*[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

*[String =] Screen.***String**

Delphi Syntax

*[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**

- **OnSendAid**
- **OnSessionState**

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

Parameters	**Description**
index	The index of the target OhioSession.

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 =] Fields.**FindByPosition(**targetPosition As TTnbOhioPosition**)**

Delphi Syntax

*[TTnbOhioField :=] Fields.***FindByPosition(**targetString:TTnbOhioPosition**)**;

Parameters

Parameters	**Description**
TargetPosition	The target row and column.

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.**FindByString(**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

Parameter	**Description**
targetString	The target string.
startPos	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).
length	The length from startPos to include in the search.
dir	An OHIO_DIRECTION value.
ignoreCase	Indicates whether the search is case sensitive. True means that case will be ignored. False means the search will be case sensitive.

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

*[Integer =] Fields.***Columns**

Delphi Syntax

*[Integer :=] Fields.***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

*[Integer =] Fields.***Count**

Delphi Syntax

*[Integer :=] Fields.***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

Parameter	Description
plane	A valid OHIO_PLANE value.

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

*[TTnbOhioPosition =] Field.***EndPos**

Delphi Syntax

*[TTnbOhioPosition :=] 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

*[Boolean =] Field.***Hidden**

Delphi Syntax

*[Boolean :=] Field.***Hidden**;

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

*[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

*[Boolean :=] Field.***Numeric**;

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

*[Boolean :=] Field.***PenSelectable**;

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]*

Remarks

All null characters and Field Attribute characters are returned as blank space characters.

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

*[Boolean=] OIA.***Alphanumeric**

Delphi Syntax

*[Boolean :=] OIA.***Alphanumeric**;

Remarks

This property returns True if the cursor is in an alphanumeric field, False in the other case.

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

*[Integer =] OIA.***MachineCheckCode**

Delphi Syntax

*[Integer :=] 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

*[Boolean =] OIA.***Numeric**

Delphi Syntax

*[Boolean :=] 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

*[Integer =] 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

Parameters	**Description**
TnbFields	The fields to refresh.
fromPos	The position where to start.

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

Constant	Value	Description
OHIO_VERSION_LEVEL	OHIO 1.00	The OHIO version level of this implementation. The form is "OHIO nn.nn"
OHIO_VENDOR_NAME	Cybele Software Inc.	The name of the vendor providing this OHIO implementation. Format is vendor defined.
OHIO_VENDOR_PRODUCT_VERSION	TN Bridge Integration Pack 3.0	The vendor product version that is providing the OHIO implementation. Format is vendor specific.

uTnbOHIO_SESSION cfg keys

Constant	Value	Description
OHIO_SESSION_TYPE	OHIO_SESSION_TYPE=	The OHIO Session Type: unknown host, 3270 or 5250 host.
OHIO_SESSION_HOST	OHIO_SESSION_HOST=	The OHIO Session Host.
OHIO_SESSION_PORT	OHIO_SESSION_PORT=	The OHIO Session Port.
OHIO_SESSION_NAME	OHIO_SESSION_NAME=	The OHIO Session Name.
OHIO_SESSION_CONFIG_RESOURCE	OHIO_SESSION_CONFIG_RESOURCE=	The OHIO Session Configuration Resource.
OHIO_SESSION_TN_ENHANCED	OHIO_SESSION_TN_ENHANCED=	
OHIO_SESSION_SCREEN_SIZE	OHIO_SESSION_SCREEN_SIZE=	The Screen Size for the OHIO Session.
OHIO_SESSION_CODE_PAGE	OHIO_SESSION_CODE_PAGE=	The Code Page for the OHIO Session.

OHIO_DIRECTION

Constant	Value	Description
OHIO_DIRECTION_FORWARD	0	Forward (beginning towards end)
OHIO_DIRECTION_BACKWARD	1	Backward (end towards beginning)

OHIO_TYPE

Constant	Value	Description
OHIO_TYPE_UNKNOWN	0	Unknown Host
OHIO_TYPE_3270	1	3270 Host

OHIO_TYPE_5250	2	5250 Host

OHIO_STATE

Constant	Value	Description
OHIO_STATE_DISCONNECTED	0	The communication link to the host is disconnected
OHIO_STATE_CONNECTED	1	The communication link to the host is connected

OHIO_PLANE

Constant	Value	Description
OHIO_PLANE_TEXT	1	Indicates Plane Text (character data)
OHIO_PLANE_COLOR	2	Indicates Plane Color (standard HLLAPI CGA color values)
OHIO_PLANE_FIELD	4	Indicates Field Attribute Plane (field attribute bytes)
OHIO_PLANE_EXTENDED	8	Indicates Extended Plane (extended attribute bytes)

OHIO_COLOR

Constant	Value	Description
OHIO_COLOR_BLACK	0	Black
OHIO_COLOR_BLUE	1	Blue
OHIO_COLOR_GREEN	2	Green
OHIO_COLOR_CYAN	3	Cyan
OHIO_COLOR_RED	4	Red
OHIO_COLOR_MAGENTA	5	Magenta
OHIO_COLOR_YELLOW	6	Yellow
OHIO_COLOR_WHITE	7	White

OHIO_EXTENDED

Constant	Value	Description
OHIO_EXTENDED_HILITE	$C0	Bitmask for Highlighting Bits
OHIO_EXTENDED_COLOR	$38	Bitmask for Color Bits
OHIO_EXTENDED_RESERVED	$07	Bitmask for Reserved Bits
OHIO_EXTENDED_HILITE_NORMAL	$00	Normal highlighting
OHIO_EXTENDED_HILITE_BLINK	$40	Blinking highlighting
OHIO_EXTENDED_HILITE_REVERSEVIDEO	$80	Reverse video highlighting
OHIO_EXTENDED_HILITE_UNDERSCORE	$C0	Under score highlighting
OHIO_EXTENDED_COLOR_DEFAULT	$00	Default color
OHIO_EXTENDED_COLOR_BLUE	$08	Blue

OHIO_EXTENDED_COLOR_RED	$10	Red
OHIO_EXTENDED_COLOR_PINK	$18	Pink
OHIO_EXTENDED_COLOR_GREEN	$20	Green
OHIO_EXTENDED_COLOR_TURQUOISE	$28	Turquoise
OHIO_EXTENDED_COLOR_YELLOW	$30	Yellow
OHIO_EXTENDED_COLOR_WHITE	$38	White
OHIO_EXTENDED_COLUMN_SEPARATOR	$02	Separator

OHIO_FIELD

Constant	Value	Description
OHIO_FIELD_ATTRIBUTE	$C0	Bitmask for field attribute
OHIO_FIELD_PROTECTED	$20	Protected field
OHIO_FIELD_NUMERIC	$10	Numeric field
OHIO_FIELD_HIDDEN	$0C	Hidden field
OHIO_FIELD_PEN_SELECTABLE	$08	Pen selectable field
OHIO_FIELD_HIGH_INTENSITY	$04	High Intensity field
OHIO_FIELD_RESERVED	$02	Reserved field
OHIO_FIELD_MODIFIED	$01	Modified field

OHIO_UPDATE

Constant	Value	Description
OHIO_UPDATE_CLIENT	0	Update initiated by client
OHIO_UPDATE_HOST	1	Update initiated by host

OHIO_OWNER

Constant	Value	Description
OHIO_OWNER_UNKNOWN	0	Unitialized
OHIO_OWNER_APP	1	Application or 5250 host
OHIO_OWNER_MYJOB	2	3270 - Myjob
OHIO_OWNER_NVT	3	NVT (Network Virtual Terminal) mode
OHIO_OWNER_UNOWNED	4	Unowned (3270 only)
OHIO_OWNER_SSCP	5	SSCP (3270 only)

OHIO_INPUTINHIBITED

Constant	Value	Description
OHIO_INPUTINHIBITED_NOTINHIBITED	0	Input not inhibited
OHIO_INPUTINHIBITED_SYSTEM_WAIT	1	Input inhibited by a System Wait state ("X SYSTEM" or "X []")
OHIO_INPUTINHIBITED_COMMCHECK	2	

OHIO_INPUTINHIBITED_PROGCHECK	3	
OHIO_INPUTINHIBITED_MACHINECHECK	4	
OHIO_INPUTINHIBITED_OTHER	5	Input inhibited by something other than above states

uTnbOHIO_SESSION_STATE

Constant	Value	Description
OHIO_SESSION_STATE_NOSESSION	0	No session established
OHIO_SESSION_STATE_SSCPLU	1	SSCPLU session state
OHIO_SESSION_STATE_LULU	2	LULU session state

uTnbOHIO_EXCEPTION

Constant	Value
OHIO_EXCEPTION_SESSIONALREADYEXISTS	Already exists a session with that name
OHIO_EXCEPTION_BADCFGFILE	Error in configuration file
OHIO_EXCEPTION_BADHOSTNAME	There is no data provided in '+OHIO_SESSION_HOST+' configuration field
OHIO_EXCEPTION_BADPORTNUMBER	Data provided in '+OHIO_SESSION_PORT+' configuration field is not a number
OHIO_EXCEPTION_INVALIDPORTNUMBER	Data provided in '+OHIO_SESSION_PORT+' configuration field is out of range
OHIO_EXCEPTION_TOOMANYSESSIONS	Too many sessions

5.2.3.2 3270 Key Values

Aid Key Constants	Value
OHIO_AID_KEY_ATTN	$0182
OHIO_AID_KEY_BACKSPACE	$0183
OHIO_AID_KEY_BACKTAB	$0184
OHIO_AID_KEY_CLEAR	$0186
OHIO_AID_KEY_CURSORSEL	$0188
OHIO_AID_KEY_DELETE	$0189
OHIO_AID_KEY_DOWN	$018B
OHIO_AID_KEY_DUP	$018C
OHIO_AID_KEY_ENTER	$018D
OHIO_AID_KEY_ERASEEOF	$018E
OHIO_AID_KEY_ERASEINPUT	$018F
OHIO_AID_KEY_HOME	$0190

OHIO_AID_KEY_LEFT	$0193
OHIO_AID_KEY_MARK	$0194
OHIO_AID_KEY_NEWLINE	$0195
OHIO_AID_KEY_PA1	$0196
OHIO_AID_KEY_PA2	$0197
OHIO_AID_KEY_PA3	$0198
OHIO_AID_KEY_PF1	$01A0
OHIO_AID_KEY_PF2	$01A1
OHIO_AID_KEY_PF3	$01A2
OHIO_AID_KEY_PF4	$01A3
OHIO_AID_KEY_PF5	$01A4
OHIO_AID_KEY_PF6	$01A5
OHIO_AID_KEY_PF7	$01A6
OHIO_AID_KEY_PF8	$01A7
OHIO_AID_KEY_PF9	$01A8
OHIO_AID_KEY_PF10	$01A9
OHIO_AID_KEY_PF11	$01AA
OHIO_AID_KEY_PF12	$01AB
OHIO_AID_KEY_PF13	$01AC
OHIO_AID_KEY_PF14	$01AD
OHIO_AID_KEY_PF15	$01AE
OHIO_AID_KEY_PF16	$01AF
OHIO_AID_KEY_PF17	$01B0
OHIO_AID_KEY_PF18	$01B1
OHIO_AID_KEY_PF19	$01B2
OHIO_AID_KEY_PF20	$01B3
OHIO_AID_KEY_PF21	$01B4
OHIO_AID_KEY_PF22	$01B5
OHIO_AID_KEY_PF23	$01B6
OHIO_AID_KEY_PF24	$01B7
OHIO_AID_KEY_RESET	$019A
OHIO_AID_KEY_RIGHT	$019B
OHIO_AID_KEY_SYSREQ	$019C
OHIO_AID_KEY_TABFORWARD	$019D
OHIO_AID_KEY_UP	$019F
OHIO_AID_KEY_PRINT	$01C0
OHIO_AID_KEY_PGUP	$01C1
OHIO_AID_KEY_PGDOWN	$01C2

5.2.3.3 5250 Key Values

Aid Key Constants	**Value**
OHIO_AID_KEY_ATTN	$0182
OHIO_AID_KEY_BACKSPACE	$0183
OHIO_AID_KEY_BACKTAB	$0184
OHIO_AID_KEY_CLEAR	$0186
OHIO_AID_KEY_CURSORSEL	$0188

OHIO_AID_KEY_DELETE	$0189
OHIO_AID_KEY_DOWN	$018B
OHIO_AID_KEY_DUP	$018C
OHIO_AID_KEY_ENTER	$018D
OHIO_AID_KEY_ERASEEOF	$018E
OHIO_AID_KEY_ERASEINPUT	$018F
OHIO_AID_KEY_HOME	$0190
OHIO_AID_KEY_LEFT	$0193
OHIO_AID_KEY_MARK	$0194
OHIO_AID_KEY_NEWLINE	$0195
OHIO_AID_KEY_PA1	$0196
OHIO_AID_KEY_PA2	$0197
OHIO_AID_KEY_PA3	$0198
OHIO_AID_KEY_PF1	$01A0
OHIO_AID_KEY_PF2	$01A1
OHIO_AID_KEY_PF3	$01A2
OHIO_AID_KEY_PF4	$01A3
OHIO_AID_KEY_PF5	$01A4
OHIO_AID_KEY_PF6	$01A5
OHIO_AID_KEY_PF7	$01A6
OHIO_AID_KEY_PF8	$01A7
OHIO_AID_KEY_PF9	$01A8
OHIO_AID_KEY_PF10	$01A9
OHIO_AID_KEY_PF11	$01AA

6 XML Reference

- **Introduction**
- **Programing Reference**

6.1 Introduction

XML (e**X**tensible **M**arkup **L**anguage) 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:

```
<TNBRIDGE>
    <Connections>
        <Connection>
            <Host ... />
            <Display ... />
        <Connection>
        .
        .
        .
    <Connections />
    <ScreenStyles>
        <ScreenStyle>
            <General>
                <Font ... />
            <General />
            <FieldMapping>
                <FieldMap>
                    <Input ... />
                    <Output ... />
                <FieldMap />
                .
                .
                .
            <FieldMapping />
        <ScreenStyle />
        .
        .
        .
    <ScreenStyles />
    <KeyboardMaps>
        <KeyboardMap>
            <KeyMap>
                <FunctionKey ... />
                <MappedKey ... />
            <KeyMap />
            .
            .
            .
        <KeyboardMap />
        .
        .
        .
    <KeyboardMaps>
      <HotSpots>
        <HotSpotsGroup>
            <HotSpot ... />
        <HotSpotsGroup />
        .
        .
        .
      <HotSpots />
      <TnbMacros>
        <Macro>
            <Action ... />
          <Macro />
          .
          .
          .
      <TnbMacros />
<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:

<Connections **MRUName**="Clemson"/>

Connection Name

Indicates connection name.

Syntax Example:

<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

<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:

<ScreenStyles **MRUName**="Green"/>

ScreenStyle Name

Indicates the screen style name.

Syntax Example:

<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:

<KeyboardMaps **MRUName**="Default++"/>

KeyboardMap Name

Specifies the keyboard map's name.

Syntax Example:

<KeyboardMap **Name**="Default++"/>

KeyboardMap Type

Indicates the keyboard map type.

Syntax Example:

<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:

<HotSpots **MRUName**="Clemson"/>

HotSpotGroup Name

Specifies the HotSpotGroup's name.

Syntax Example:

<HotspotGroup **Name**="Clemson"/>

HotSpotGroup StartCol

Specifies the HotSpotGroup's start column number coordinate.

Syntax Example:

<HotspotGroup **StartCol**="0"/>

HotSpotGroup StartRow

Specifies the HotSpotGroup's start row number coordinate.

Syntax Example:

<HotspotGroup **StartRow**="0"/>

HotSpotGroup EndCol

Specifies the HotSpotGroup's end column number coordinate.

Syntax Example:

<HotspotGroup **EndCol**="0"/>

HotSpotGroup EndRow

Specifies the HotSpotGroup's end row number coordinate.

Syntax Example:

<HotspotGroup **EndRow**="0"/>

HotSpotGroup ViewAs

Indicates the HotSpotGroup's shape. This shape can be: None, Plain, Button or Link.

Syntax Example:

<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:

<macro **version**="1"/>

Macro Name

Specifies the Macro's name.

Syntax Example:

<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 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" />
                          <MappedKey LShift="0" RShift="0" LControl="0" RControl="0"
LMenu="0" RMenu="0" NumLock="0" ScanCode="69" />
                   </KeyMap>
                   .
                   .
                   .
            </KeyboardMap>
            .
            .
            .
     </KeyboardMaps>
     <Hotspots MRUName="Clemson">
            <HotspotGroup Name="Clemson" StartRow="0" StartCol="0" EndRow="0"
EndCol="0" ViewAs="">
                   <Hotspot Name="Clemson University" Id="{2000374A-DAF7-498A-9E6D-
FCB10E98E671}" StartRow="1" StartCol="32" EndRow="2" EndCol="50" ViewAs="Button"
ActionFieldData="@E" Pattern="Clemson University" FgColor="Navy" BgColor="Black" />
                   .
                   .
                   .
            </HotspotGroup>
            .
            .
            .
     </Hotspots>
</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*

*****_Find the help file for TN Bridge Host Integration Pack for DotNET here: http://www.cybelesoft.com/helps/tnb/dotNet/index.html_

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:

  http://www.cybelesoft.com/buy/

- 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/

- 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:

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):**

```
 new Tnblib.TnbXLicense().Key = "Y2Ja9EstTBt+9hNtHH28fDoVSWOLT7QV"+
                                "igoceLrR0UvY3T+Kh-gyi4Gpd21BvQZh"+
                                "5jDd6TbljNIKAFEpR9b5UcekvaEoouDz"+
                                "7iF9lQTIg-Q0HaeICm9OFznRM0xpSfnQ"+
                                "m0f0ZhAH7dmPL9eytIbMlMaNn+xdCs3B"+
                                "VfmGuzZgs6n9wvScvjD3E6UDNUuhFKZ9"+
                                "lOuTc3H6M3YZbwIrb7gH5lmw2EvhHMSn"+
                                "B5TiZkxOoSP-6SYJ5ZoxiiL5er7jOjZw"+
                                "36T7GfN3amk94YndwDfJpKOCOFOuSgex"+
                                "ek8Til9bjfNfehnEcM7u5RE2arYAganL"+
                                "qRuL3XY-ktdvnp1CUBwvJ+";
```

- **VB/VB.NET example on how to embed the code (replace with you own license):**

```
Dim License As Tnblib.TnbXLicense

License = New Tnblib.TnbXLicense()
License.Key = "Y2Ja9EstTBt+9hNtHH28fDoVSWOLT7QV" + _
         "igoceLrR0UvY3T+Kh-gyi4Gpd21BvQZh" + _
         "5jDd6TbljNIKAFEpR9a5LcekvaEoouDz" + _
         "7iF9lQTIg-Q0HaeICm9OFznRM0xpSfnQ" + _
         "m0f0ZhAH7dmPL9eytIbMlMaNn+xdCs3B" + _
         "VfmGuzZgs6n9wvScvjD3E6UDNUuhFKZ9" + _
         "lOuTc3H6M3YZbwIrb7gH5lmw2EvhHMSn" + _
         "B5TiZkxOoSP-6SYJ5ZoxiiL5er7jOjZw" + _
         "36T7GfN3amk94YndwDfJpKOCOFOuSgex" + _
         "ek8Til9bjfNfehnEcM7u5RE2arYAganL" + _
         "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:

```
  implementation
  uses uTnbIPack;
  ...
  ...
initialization
  LicenseKey := 'Y2Ja9EstTBt+9hNtHH28fDoVSWOLT7QV'+
                'igoceLrR0UvY3T+Kh-gyi4Gpd21BvQZh'+
                '5jDd6TbljNIKAFEpR9a5UcekvaEoouDz'+
                '7iF9lQTIg-Q0HaeICm7OFznRM0xpSfnQ'+
                'm0f0ZhAH7dmPL9eytIbMlMaNn+xdCs3B'+
                'VfmGuzZgs6n9wvScvjD3E6UDNUuhFKZ9'+
                'lOuTc3H6M3YZbwIrb7gH5lmw2EvhHMSn'+
                'B5TiZkxOoSP-6SYJ5ZoxiiL5er7jOjZw'+
                '36T7GfN3amk94YndwDfJpKOCOFOuSgex'+
                'ek8Til9bjfNfehnEcM7u5RE2arYAganL'+
                '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:

```
implememtation
uses uTnbIPack;
...
...
```

```
initialization
LicenseKey := 'Y2Ja9EstTBt+9hNtHH28fDoVSWOLT7QV'+
'igoceLrR0UvY3T+Kh-gyi4Gpd21BvQZh'+
'5jDd6TbljNIKAFEpR9a5UcekvaEoouDz'+
'7iF9lQTIg-Q0HaeICm9OFznRM0xpSfnQ'+
'm0f0ZhAH7dmPL9eytIbMlMaNn+xdCs3B'+
'VfmGuzZgs6n9wvScvjD3E6UDNUuhFKZ9'+
'lOuTc3H6M3YZbwIrb7gH5lmw2EvhHMSn'+
'B5TiZkxOoSP-6SYJ5ZoxiiL5er7jOjZw'+
'36T7GfN3amk94YndwDfJpKOCOFOuSgex'+
'ek8Til9bjfNfehnEcM7u5RE2arYAganL'+
'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 and we will write you back timely. You can also contact us through Live Chat by pressing this icon in our website:

and immediately have a conversation with a representative without even having to pick up the phone.

Cybele Software Inc.
3422 Old Capitol Trail, suite 1125
Wilmington, DE - 19808
Phone: (302) 892-9625
Fax: (302) 295-9995
e-mail: support@cybelesoft.com
http://www.cybelesoft.com