x3270 for Borland Delphi
Version 2.0
Release Notes

July 14, 1995

This file contains some informations about x3270 for Borland Delphi release 2.0. This informations are provided as documentation of *demo version*.

Using Write to View This Document

If you enlarge the Write window to its maximum size, this document will be easier to read. To do so, click the Maximize button in the upper right-hand corner of the window. Or, click on the system menu in the upper left-hand corner of the Write window (or press Alt+Spacebar), and then choose the Maximize command.

To move through the document, press Page Up or Page Down or click the arrow at the top or bottom of the scroll bar along the right side of the Write window.

To print the document, choose the Print command from the File menu.
For help using Write, press F1.
To read other on-line documents, choose the Open command from the File menu.

Contents

This document contains informations on the following topics:

Read Me First! _____

x3270 is the first Unit Pascal file that allows you to interface applications written by Borland Delphi compiler with the world of mainframe across one or more sessions of IBM 3270 emulation.

The unit services guarantee the full compatibility with all emulators software such as Access by Eicon or MW3270 by Olivetti, they are on lan or on cluster.

x3270 provides very important development tools for Delphi developers and supply base and advanced services how ***connecting to mainfraime, sending a keystroke to current presentation space, searching a string on presentation space, copy data from host*** and many others.

All the functionalities are implemented as pascal class (T3270 class), so all host informations such as shortname, longname and many others are encapsulated in the object.
You can instantiate many x3270 objects that are related to different host sessions and work with them *simultaneously* by sending messages.

API session types are referred to as presentation space. Presentation spaces are regions in computer storage that can be displayed in one of the windows on the IBM 3270 Workstation screen. This includes the host system, PC, and notepad physical sessions, but it also includes connections to the workstation control mode supervisor (WsCtrl).

API applications are valuable in a variety of work environments. They can increase productivity for experienced users and provide a shorter Teaming curve for inexperienced users.

The following are some advantages of using T3270.

make the IBM 3270 WorkStation Program easier to use by:

. Automating repetitive tasks
. Masking from the user any complicated, application-generated screens, commands, and the like.
. Automating the logon sequence
. Simplifying IBM 3270 Workstation setup (creating windows, sizing windows, developing notepad functions, and so forth)
. Simplifying complex IBM 3270 Workstation tasks (copy, file transfer, to name a few).
. simplify existing host system applications.

provide unattended operation, a programmed operator that monitors IBM 3270 Workstation tasks without human intervention. This programmed operator can:

. Monitor events that are serial in nature (for example, data center applications)
. Automate console operation
. Monitor response time and availability
. Do stress testing.

let you *create composite screen applications using input from more than one host system.* The input for a composite screen is formatted and presented within a PC presentation space. The user sees a single screen image of information that would normally require multiple host system computer sessions or, before the IBM 3270 Workstation, multiple terminals.

let you *write programs that divide work between a host system and the IBM 3270 Workstation*. This allows an application to access files and programs at either the host system or the PC session. The more trivial tasks (file editing, simple graphics, etc.) can be done in the PC session and the more powerful host system can do data base searches, updates, and retrievals when needed. The terminal operator does not even need to know whether a PC or host system application is doing the task. The goal is simply to accomplish the task.

Getting Started!

To write an application program, you'll need the following:

1. *Borland Delphi compiler for Microsoft Windows*

2. *EHLLAPI.DLL, the interface module lets your program communicate with the standard IBM Low-Level Application Program Interface (included in your 3270 emulation software).*

To write an effective API program, you should think of it as a programmed terminal operator. When terminal operators sit down at a work station, they typically look at the screen, determine "where they are" in the application and take the necessary actions to get to the desired starting point.

This may involve an action as simple as pressing the *Clear key*, or it may require routing through a VTAM network to the proper application.

When you write your application program, your "programmed operator" (the software "user" that performs and monitors activities in your IBM 3270 Workstation without actual human intervention) performs these user functions.

Before running your application, you must load the emulation program with APIs support.
That's all! Very easy, what you think?

Tecnichal Reference *(small subset)* _____

Listed methods and instance variables below are all the functionalities that you can use in your Delphi applications for performing host connections. For more convenience there are some brief explanations for only a small subset of this!

T3270 Class!

Status: TStatus;

constructor *T3270.Create(const cSN: string);*

function T3270.Disconnect: Boolean;
function T3270.SendKey(cKeyStroke: string): Boolean;
function T3270.CursorAt(siPos: Integer): Boolean;
function T3270.CursorPosition: Integer;
function T3270.CopyScreen: pChar;
function T3270.CopyScreenAttr: pChar;
function T3270.ReadAt(const nPos, nBytes: Integer): string;
function T3270.WriteAt(const cStringa: string; const nPos: Integer):Boolean;
function T3270.SearchAt(const cStringa: string; const nPos: Integer): Boolean;
function T3270.SearchString(const cStringa: string): Integer;
function T3270.ConvertRowCol(const nRiga,nColonna: Integer): Integer;
function T3270.FieldAttribute: Integer;
function T3270.FieldCharacter: Boolean;
function T3270.FieldNumeric: Boolean;
function T3270.FieldProtected: Boolean;
function T3270.FieldUnProtected: Boolean;
function T3270.FieldUpdated: Boolean;
function T3270.FieldHiColor: Boolean;
function T3270.FieldLowColor: Boolean;
function T3270.FieldLength: Integer;
function T3270.FieldGet: string;
function T3270.FieldLocate(const cTipo: string; const nPos: Integer): Integer;
function T3270.FieldPut(const cStringa: string): Boolean;
function T3270.FieldAttributeAt(const nPos: Integer): Integer;
function T3270.FieldCharacterAt(const nPos: Integer): Boolean;
function T3270.FieldNumericAt(const nPos: Integer): Boolean;
function T3270.FieldProtectedAt(const nPos: Integer): Boolean;
function T3270.FieldUnProtectedAt(const nPos: Integer): Boolean;
function T3270.FieldUpdatedAt(const nPos: Integer): Boolean;
function T3270.FieldHiColorAt(const nPos: Integer): Boolean;
function T3270.FieldLowColorAt(const nPos: Integer): Boolean;
function T3270.FieldLengthAt(const nPos: Integer): Integer;
function T3270.FieldGetAt(const nPos: Integer): string;
function T3270.FieldPutAt(const cStringa: string; const nPos: Integer): Boolean;
function T3270.FieldSkip: Boolean;
function T3270.LockKeyboard: Boolean;
function T3270.UnLockKeyboard: Boolean;
function T3270.OperInfoArea: string;
function T3270.CapsLock: Boolean;
function T3270.InsertMode: Boolean;
function T3270.Row: Integer;
function T3270.Col: Integer;

procedure T3270.ShowResult;
procedure T3270.Free;

property T3270.ShortName: Char;
property T3270.LongName: string;

property T3270.SessionType: string;
property T3270.Rows: Integer;
property T3270.Columns: Integer;
property T3270.SizeOfPS: Integer;
property T3270.InitOK: Boolean;

Environment Functions/Properties

SysNumber: Integer;
SysDesc: string;

function HQuerySessions: string;
function HSetParam(const cParam: string): Boolean;
function HResetSystem: Boolean;

T3270.SendKey;

SendKey() sends a string of keystrokes to the connected presentation space. The session must have the keyboard unlocked for input before accepting keystrokes.You define the strings of keystrokes to be sent with the calling data string parameter.

The keystrokes appear to the target session as though they were entered by the terminal operator. You can also send all attention identifier (AID) keys such as Enter, PA1, and so forth. All the fields that are protected for input or are numeric only must be treated accordingly.

T3270.WriteAt;

WriteAt() copy an ASCII data string directly into the current connected presentation space at the location specified by the calling parameter (default: current cursor position).

T3270.SearchString;

SearchStringS() lets your HLLAPI program examine the current connected presentation space for the occurrence of a specified string. If the string is not located, 0 is the return code. If the string is found, then the return code is set to the string's beginning location in the presentation space.

This location represents a position in the presentation space based on the layout where the upper left corner (row 1, column 1) is location 1 and the bottom right location is *2000* for PCs and alternate presentation spaces without 3270 Keystroke Emulation, *1920* for Model 2s, notepads, and alternate presentation spaces with 3270 Keystroke Emulation, *2560* for Model 3s; *3440* for Model 4s and *3564* for Model 5s.

This method is useful in determining when a specific 3270 presentation space is available. If your programmed operator is expecting a specific prompt or message before sending data, SearchString() allows you to check for the prompt message before continuing.

T3270.SearchAt;

SearchStringS() lets your HLLAPI program examine the current connected presentation space for the occurrence of a specified string at specified position. If the string is not located, 0 is the return code. If the string is found, then the return code is set to the string's beginning

location in the presentation space. For more informations, see SearchString() method.

T3270.CopyScreen;

The CopyScreen() method is used to copy all the current connected presentation space into an application buffer. The offset of the string into the presentation space is based on a layout in which the upper left comer (row 1/column 1) is location 1 and the bottom right comer is the maximum screen size for the (for a Model 2, for example, this would be 1920.).

T3270.CopyScreenAttr;

The CopyScreeAttr() method is used to copy all the current connected presentation space into an application buffer with all screen attributes.

T3270.CursorPosition;

This method indicates the position of the cursor in the current connected presentation space by returning the cursor position.

T3270.CursorAt;

This method move host cursor at specified position in the current connected presentation space.

T3270.FieldLocate;

This method examines a field within the connected presentation space for a specified occurrence. If the target field is found, this method returns the decimal position numbered from the beginning of the presentation space. (For example, the "row 1, column 1" position is numbered "1".)

FieldLocate() can be used to search for either protected or unprotected fields but only in a field-formatted presentation space. This means that the target field can be in a host system presentation space

---

T3270.FieldGet;

This method return the contents of field within the connected presentation space at the current cursor position and return the value as string.

T3270.FieldPut;

This method assign a value (string) on field within the connected presentation space at the current cursor position.

T3270.FieldLength;

This method return the length of field within the connected presentation space at the current cursor position.

T3270.FieldAttribute;

This method return the attribute value (numeric) of the field within the connected presentation space at the current cursor position.

T3270.Disconnect;

Disconnect drops the connection between your HLLAPI application program and the current connected presentation space or WsCtrl.

T3270.Rows

Number of rows on the current connected presentation space

T3270.Columns

Number of columns on the current connected presentation space

---

T3270.ShortName

The session shortname of the current connected presentation space

---

T3270.LongName

The session longname of the current connected presentation space

---

T3270.Status

It contains an internal object that describe the status after calling API methods. x3270 has an internal dictionary that contains all standard IBM return code and related explanations. So, after calling any method that interact with low-level subsystem, you can verify return code in the ***Number*** property T3270.and read a ***Description*** property T3270.of ***Status*** object. However, if an error occurred, there is a method, ShowResult(), that use the instance variable Status and shows it in a dialog box.

Evaluation Version Files

This evaluation version contains x3270.DCU file and xDEMO.PAS file that supply an host application framework (look up carefully!).

The only difference between this evaluation version and the full product is that the Unit file have 50 fixed number calls to functions/methods, making applications unsuitable for general distribution.

We hope this will provide developers with an excellent opportunity to thoroughly evaluate our products before purchasing.

x3270 for CA-Clipper

For all users that are working with CA-Clipper compiler, *there is x3270 for CA-Clipper.* The most interesting characteristic is the possibility to interface the 286 DOS Extenders CA-Exospace and Blinker v3.01 that allows CA-Clipper applications address up to 16 megabyte of memory.

It's now possible to write very complex downsizing procedures that operate in very tight memory situations (by only 200k of free RAM, 800k of executable code could be activated, code that exchange continually data among Host and PC).

The package include libraries EXO3270 and BLI3270 and supports DUAL MODE capability offered by Blinker. So, you can write only one program that can be executed in both real mode and protected mode.

In the Clipper forum there is a DEMO version (x3270.zip) of x3270 v1.0 for CA-Clipper

The actual release on Compuserve is a DEMO release that allows you to use some of the available services in the commercial release with a maximum limit of 10 host calls, sufficient however for testing all the functionality offered by the library.

The files included in x3270.ZIP are the followings:

* READ.ME -> General Informations
* x3270.CH -> Include file of keystrokes definitions
* BLI3270.LIB -> Release for Blinker 3.01 (Dual Mode Compatible)
* EXO3270.LIB -> Release for CA-Exospace v1.0f (Protected Mode Only)
* EXAMPLE.PRG -> Sample of application

x3270 for CA-Visual Objects

For all users that are working with CA-Visual Objects compiler, *there is x3270 for CA-Visual Objects.* Either products (VO and Delphi) are full object oriented! This package include x3270.DLL file, the prototyping x3270.AEF file that you must import into your CA-Visual Objects repository and xDEMO.AEF file that supply an host application framework, such us xDEMO.PAS for Delphi developers.

For testing the products, please see 3d Party News on Visual Objects Compuserve Forum, file x3270-VO.zip

For other informations about x3270 (all versions), please contact:

. Author:

Mauro Tronto
Via Jesi, 43
60127 ANCONA (ITALY)

Home: (39)-71-890604
CompuServe ID: 100103,2660
Internet: 100103.2660@compuserve.com