VISUAL NPL DEVELOPER'S GUIDE

Version 2.0

2nd Edition - April 1997

23600 N. Milwaukee Avenue
Vernon Hills, IL 60061
U.S.A.

PHONE: (847) 634-8700
FAX: (847) 634-8718
E-MAIL: sales@niakwa.com or support@niakwa.com

Contents

Documentation Conventions

This manual uses the following typographic conventions to describe NPL code and constructs:

Example of NPL Conventions	Description
`“VnplDev”`	Names of NPL modules (programs) are enclosed within quotations in this font.
`'VnClose, 'VnCmd`	Names of NPL functions and procedures appear in this font, preceded by an apostrophe.
`_VnSys, VnPrint$`	Names of NPL constants and variables appear in this font.

`PROCEDURE 'NextRecord/PUBLIC`
`  ;NPL`
`  ;`
`END PROCEDURE 'NextRecord`
	Code examples written in NPL appear in this font. When in proximity to VB code, the NPL code will contain anembedded `;NPL` as an explanatory `REMark` statement.

These typographic conventions describe Visual Basic code and constructs:

Example of VB Conventions	Description
Main, VnDevDef, VnSetCtrl	Names of VB methods, events, functions and procedures appear in bold.
MyProject, Form1, VnplLink, VnplUtil	Programmatic names (not filenames, titles, or captions) of VB objects appear in bold and italics.
BackColor, Tag, Visible, Clipboard, Printer, Screen	Names of VB properties and system objects appear with initial letter(s) capitalized.
File, Edit, Tools, Next, Previous, OK, Cancel	Text titles and captions of VB menu choices, tab choices, buttons, check boxes and other objects appear in italics.
`Private Sub NextBtn_Click()`	Code examples written in VB appear in this

`  Rem VB`
`  VnCallProc "NextRecord"`
`End Sub`
	font with `Rem VB` embedded within the code as an explanatory `Remark` statement.

These other typographic conventions also appear in the manual:

Example of Other Conventions	Description
BOOT.OBJ, DEMOS.NPL, SETUP.EXE, .DLL	Filenames of native Windows or DOS files (including NPL diskimage filenames), filename extensions, paths and DOS command lines appear in this bold font.
F5, TAB, DEL	Names of keys and key sequences appear in small capital letters.
Properties, methods, events	In text, italic letters indicate defined terms, usually the first time they occur in the book. Italics also are used occasionally for emphasis.
RETURN	Text you're instructed to type in appears in this font.

Other syntactical conventions for the NPL language appear as described in the Introduction to *NPL Technical Reference Guide, Statements Guide*.

CHAPTER 1

Introduction

This chapter gives a general introduction to Visual NPL. It includes the following:

- A description of Visual NPL
- System requirements for Visual NPL
- Instructions for installing Visual NPL
- A listing of the directories and files created by the installation program
- A discussion of what’s new in version 2.0
- Instructions for upgrading from version 1.0 to version 2.0.

1.1 What is Visual NPL?

Visual NPL is a development tool that provides a flexible, interactive link between Niakwa Programming Language (NPL) applications and Microsoft Visual Basic (VB) applications. It has been designed so that developers can use existing NPL code as a logic engine and use Visual Basic forms as a user interface engine to create state-of-the-art Windows programs.

Visual Basic is a graphical user interface design tool that works by creating forms that are controlled by code written in the Visual Basic for Applications (VBA) language. Visual NPL creates a link between NPL and VB so that NPL code can be used to control the VB forms. This allows the NPL developer to add a fully functional Windows user interface to proven NPL code to perform the underlying logic functions of a program.

As a benefit, Visual NPL developers can use any of the controls available in the third-party OLE control (OCX) marketplace. These controls do everything from drawing dice to providing full spreadsheet or word-processing capabilities. There are even controls that provide access to the many facets of the Internet.

Visual NPL applications consist of the following parts:

- A VB program consisting of several forms and a small amount of VBA code to transfer control to the NPL program
- The VB form, VBA code, and OCX that facilitate communications with the NPL program
- An NPL program that performs the underlying logic functions of the application and controls the VB forms
- An NPL external, dynamic-link library (DLL) and disk image that facilitate communications with the VB program.

The code required on the VB side is remarkably simple and requires very little knowledge of the VBA language. However, in order to create professional Windows programs, it is necessary to become fairly knowledgeable about the Visual Basic form-design tools. Fortunately, this feature of Visual Basic is easy to learn and understand.

1.2 System Requirements

This section discusses what you need in order to use Visual NPL.

1.2.1 Developer Knowledge

It is assumed throughout this manual that the developer has a good understanding of Microsoft Windows. In order to develop polished user interfaces in Windows, it is necessary to be familiar with many of the features available in this environment. It is not, however, necessary to know anything about Windows programming in the traditional sense. The Visual Basic environment hides the many complexities of native Windows programming and presents a greatly simplified forms-based approach.

It is also assumed that the developer has a good understanding of the NPL Release IV features. Visual NPL makes heavy use of modules, long variable names, functions and procedures, structured programming techniques, and nearly every other Release IV feature.

1.2.2 Hardware

Although there are no specific hardware requirements for using Visual NPL over and above those of NPL and VB, you may want to add or change your existing hardware to achieve the best results in the Visual NPL environment. For example, you may want to upgrade your video monitor for doing form design. This is a screen-intensive activity; therefore, the bigger the monitor, the better. In addition, although 15-inch monitors are adequate, 17-inch monitors really make the Visual Basic environment shine. Furthermore, a fast video adapter can make a significant difference, even surpassing performance gains derived from a processor upgrade, for doing highly graphical work.

1.2.3 Software

To develop applications in Visual NPL, you need the following:

- MS Windows version 3.1 or later (Windows 95 recommended)
- NPL for MS-Windows Release 4.2 or later
- Microsoft Visual Basic version 4.0 (the 16-bit version of any edition)

Note You must use the 16-bit version of Visual Basic. All editions of Visual Basic 4.0 (Standard, Professional, and Enterprise) come with both the 16-bit and 32-bit versions. When installing Visual Basic, make sure to select the 16-bit version when you are prompted to choose between the two.

1.3 Installation

To install Visual NPL from the distribution diskette on your hard disk drive, run the **SETUP.EXE** program on the diskette. This program does the following:

- Prompt you for the drive and directory into which to copy the Visual NPL files
- Decompress the files and copy them to the specified directory
- In Windows 3.1, create a new program group (or in Windows 95, a new folder) named "Visual NPL 2.0"
- Register the OCX used by Visual NPL within Visual Basic
- Create icons for the **README.TXT** file and the demo programs

Important If you have the 32-bit and the 16-bit versions of Visual Basic installed, you may have to modify the file name associated with the icon for the demo programs. This is because **SETUP.EXE** associates a file name with the icon as follows:

```
DEMOS.VBP
```

This is appropriate for most installations because VB registers itself as the handler for files with the **VBP** extension (that is, VB project files). Therefore, when the user double-clicks an icon, Windows looks in the registry to determine which program to use to open the corresponding file. Unfortunately, if the 32-bit version of Visual Basic is installed, that version may be registered as the handler for **VBP** files. In this case, the projects will fail to open properly unless you change the command line for the icon so that it includes the full path of the 16-bit VB executable, as follows:

```
C:\VB\VB.EXE  DEMOS.VBP
```

1.4 File List

Visual NPL consists of this Guide and one disk with a compressed setup program and files. Running **SETUP** decompresses the files into:

Destination Directory

File Name	Description
README.TXT	Last-minute information not available as of this printing
VNPL.NPL	Main Visual NPL disk image
VNPLCHAR.FRM	VB form used for creating controls-on-the-fly
VNPLCHAR.FRX	Graphics file used in VNPLCHAR.FRM
VNPLDEV.BAS	Developer-modifiable VB code
VNPLLINK.FRM	VB form for communicating with NPL
VNPLUTIL.BAS	Main VB routines and constants

Subdirectory HELLO_E

File Name	Description
BOOT.OBJ	NPL boot program
HELLO.EXE	Compiled program
HELLO.FRM	Main VB form
HELLO.FRX	Graphics file used in HELLO.FRM
HELLO.NPL	NPL demo program code
HELLO.VBP	VB project file
VNPLDEV.BAS	Developer-modifiable VB code

Subdirectory HELLO_C

File Name	Description
BOOT.OBJ	NPL boot program
HELLO.EXE	Compiled program
HELLO.FRM	Main VB form
HELLO.FRX	Graphics file used in HELLO.FRM
HELLO.NPL	NPL demo program code
HELLO.VBP	VB project file
VNPLDEV.BAS	Developer-modifiable VB code

Subdirectory DEMOS

File Name	Description
`BOOT.OBJ`	NPL boot program
`COLORS.FRM`	Form used in the “Colors” demo
`COLORS.FRX`	Graphics used in`COLORS.FRM`
`COMBOX.FRM`	Form used in the “Combo Box” demo
`COMBOX.FRX`	Graphics used in`COMBOX.FRM`
`CTRLVAL.FRM`	Form used in the “Validation - Control” demo
`CTRLVAL.FRX`	Graphics used in`CTRLVAL.FRM`
`CUSTDB.FRM`	Form used in the “Database Access” demo
`CUSTDB.FRX`	Graphics used in`CUSTDB.FRM`
`CUSTDB.LDB`	Database used in the “Database Access” demo
`CUSTDB.MDB`	Database used in the “Database Access” demo
`CUSTREC.FRM`	Form used in the “Customer Record I/O” demo
`CUSTREC.FRX`	Graphics file used in`CUSTREC.FRM`
`DEMOS.EXE`	Compiled program
`DEMOS.NPL`	NPL code for all demos
`DEMOS.VBP`	VB project file
`GRID.FRM`	Form used in the “Unbound Data Grid” demo
`GRID.FRX`	Graphics used in`GRID.FRM`
`IMGVIEW.FRM`	Form used in the “Image Viewer” demo
`IMGVIEW.FRX`	Graphics file used in`IMGVIEW.FRM`
`INSPECT.FRM`	Form used in the “Objects - Inspect” demo
`INSPECT.FRX`	Graphics file used in`INSPECT.FRM`
`KEYVAL.FRM`	Form used in the “Validation - Keystroke” demo
`KEYVAL.FRX`	Graphics file used in`KEYVAL.FRM`
`MAINFORM.FRM`	Main VB form, Common Dialog, and Boxes demos
`MAINFORM.FRX`	Graphics used in`MAINFORM.FRM`
`POSTVAL.FRM`	Form used in the “Validation - Post” demo
`POSTVAL.FRX`	Graphics used in`POSTVAL.FRM`
`PROGRESS.FRM`	Form used in the “Progress” demo
`PROGRESS.FRX`	Graphics used in`PROGRESS.FRM`
`PRTSETUP.FRM`	Form used in the “Printer - Setup” demo
`PRTSETUP.FRX`	Graphics used in`PRTSETUP.FRM`

REALVAL.FRM	Form used in the “Validation - Real-Time” demo
REALVAL.FRX	Graphics used in **REALVAL.FRM**
SETPROPS.FRM	Form used in the “Objects - Set Properties” demo
SETPROPS.FRX	Graphics used in **SETPROPS.FRM**
SWOOSH.BMP	Niakwa logo as a bitmap
SWOOSH.ICO	Niakwa logo as an icon
VNPLDEV.BAS	Developer-modifiable VB code

\Windows\System Directory of the Destination Drive

File Name	Description
VNPL16.DLL	NPL external library
VNCON16.OCX	VB connection control
VNPL.LIC	License file for **VNCON16.OCX**
CTL3DV2.DLL	3–D look and feel support
MFC250.DLL	Microsoft Foundation Classes
MFCO250.DLL	Microsoft Foundation Classes for OLE
TDBGS16.OCX	Updated VB data bound grid control
GRDKRN16.DLL	Support DLL for **TDBGS16.OCX**
REGSVR.EXE	Program for registering OCXs

1.5 What’s New in Version 2.0?

Visual NPL 2.0 contains many new features as well as several enhancements to the features found in version 1.0. The following are the most significant new features:

- Event-driven programming is completely supported.
- Any method of any object can now be called from NPL.
- NPL procedures can now be called from VB.
- VB object variables can now be accessed and treated as variables in NPL.
- Support is provided for Visual Basic 4.0 and OLE.
- Font translation for foreign languages is provided.

The following are the most significant enhancements:

- Improved performance on the most commonly used operations
- Full property access, such that any type of property (not just numbers and strings) can now be retrieved or set from NPL
- Increased ability to create user-defined VB commands
- Revised controls-on-the-fly code that is faster, more flexible, and more stable
- Expanded demo programs that provide better coverage of the new and existing features
- A simplified file structure that combines many files into one

1.6 Moving from Version 1.0 to 2.0

Upgrading from version 1.0 to version 2.0 involves several steps that must be performed in a specific order. Start by making a back-up copy of all files. Then you need to make changes to your VB and NPL programs as described in the following procedures.

⇔ **To upgrade your VB program**

1. Open the existing project with the 16-bit version of VB 4.0.
2. Select *Yes* if asked to update any custom controls.
3. Select *Ok For All* when prompted to save files in VB 4.0 format.
4. Select *Don't Add* when prompted about the DAO library.
5. Remove the ***VnplLink*** form from your project.
6. Remove the ***VnplUtil*** module from your project.
7. Remove **`VNPLCTRL.VBX`** from your project.
8. Add ***Vnpl Connection Control*** to your project.
9. Add the new ***VnplLink*** form to your project.
10. Add the new ***VnplUtil*** module to your project.
11. Copy the new **`VNPLDEV.BAS`** file to your project directory and add it to your project to create the ***VnplDev*** module.
12. Copy any changes you made to the **Main** procedure in the ***VNPLMAIN*** module to the **Main** procedure in the ***VnplDev*** module.
13. Copy any changes you made to the **VnSetForm** function in the ***VNPLMAIN*** module to the **VnSetObj** function in the ***VnplDev*** module (note the additional and changed parameters).

14. Copy any changes you made to the **VnSetCtrl** function in the ***VNPLMAIN*** module to the **VnSetCtrl** function in the ***VnplDev*** module (note the additional and changed parameters).
15. Copy any changes you made to the **VnDevDef** function in the ***VNPLMAIN*** module to the **VnDevDef** function in the ***VnplDev*** module (note the additional and changed parameters).
16. Remove the ***VNPLMAIN*** module from your project.
17. Delete the **`VNPLMAIN.BAS`** file in your project directory.
18. Save your project.

⇔ **To upgrade your NPL program**

1. Change the boot program so that the device table specifies the new **`VNPL.NPL`** diskimage.
2. Copy the `"VnplDev"` module from the **`VNPL.NPL`** diskimage to your main diskimage under the name `"VnplDev2"`.
3. Copy any changes you made to the `"VnplDev"`, `"VnplErr"`, and `"VnplCtrl"` modules to the `"VnplDev2"` module.
4. Delete the `"VnplDev"`, `"VnplErr"`, and `"VnplCtrl"` modules.
5. Rename the `"VnplDev2"` module to `"VnplDev"`.

(this page blank)

C H A P T E R 2

Visual Basic Fundamentals

This chapter is an introduction to the Visual Basic programming environment. It covers the following:

- An introduction to Visual Basic and to object-based development
- A discussion about creating and using Visual Basic projects
- A description of how to create and use forms and controls
- Information about various Visual Basic language issues

2.1 About Objects

In simplest terms, an object is a piece of data and the code that manipulates the data. By combining these two programming elements into a single entity, objects give developers a powerful toolset to use in building applications. Visual NPL bridges the gap between the procedural world of traditional NPL programming and that of Windows objects available through VB. These are characteristics of objects:

- New objects can be created and destroyed dynamically.
- Implementation details of an object are hidden from programs using the object.
- Access to an object's data is through functions provided by the object's code.
- Permanent objects have functions to store and retrieve their data.
- Graphical objects have functions to draw the object and control its user interface.

An object is defined by the organization of its pieces—that is, of its code and data. Obviously, for this approach to work, there must be a standard that explains how to access and use objects in general. Without this standard, all you have is a some code that manipulates a data structure using a proprietary interface. The evolution of object standards is described in the next section.

2.1.1 Windows Objects

The mechanism used to create objects in Windows is the dynamic-link library (DLL). A DLL is the same as an EXE except that there is no "main" function. Instead, there is a group of functions (a library) that can be called from any program. Programs load the library when they need to use it, and unload it when done (called *dynamic linking*).

Unfortunately, there is no way to tell what's in a DLL. This is where VBXs (VB controls) come in. A VBX is a DLL that conforms to the standard set by the original VB development environment, which specifies a set of data structures used to describe an object and a set of functions used to get that data. All VBXs must contain these functions. Furthermore, the standard organizes objects so that they consist of a set of *properties* (the object's data), *methods* (the operations that can be performed on the data), and *events* (notifications sent from the object to VB).

There were three problems with the VBX standard:

- It wasn't an open standard because it depended on Visual Basic.
- It wasn't portable to the 32-bit environment.
- It was limited and needed to have additional functionality.

In light of these limitations, Microsoft created the OCX (or OLE control) standard. This is an open, 16-bit and 32-bit standard based on Microsoft OLE. In fact, OLE was given new functionality in order to support the approach based on properties, methods, and events used in the VBX standard. The OCX standard has since been enhanced to accommodate the Internet and has been renamed to ActiveX

An advantage of this approach is that it provides programmatic access to any OCXs resident on your computer, including those embedded in other applications, such as Microsoft Word or Netscape Navigator.

2.1.2 VB Objects

Visual Basic version 3.0 and earlier versions supported the VBX standard. VB 4.0 supports both VBXs and 16-bit OCXs in its 16-bit version and only 32-bit OCXs in its 32-bit version. In all cases, the programming interface is the same.

A VB program displays a set of forms as its windows. The items on these forms are called controls. Both forms and controls are objects. VB provides the following built-in objects (not in OCXs):

- Forms
- Standard Windows controls (buttons, text fields, labels, etc.)
- List controls (lists, drop-down lists, and grid controls)
- Graphics display (images and pictures)
- Disk-access controls (drive, directory, and file lists)
- 3-D effects (lines and panels)
- Database-access controls
- Windows common dialog-box controls (for file opening, font selection, printer setup, and so on)
- Communications (serial port) control

In addition, Visual Basic provides the following system objects:

App	Application
Clipboard	Clipboard
Err	Last general error
Error	Last data access error
Forms	List of currently loaded forms
Printer	Current printer
Printers	List of available printers
Screen	Screen

There are also severalthird-party OCXs, some provided with VB and others available from the companies that developed them.

2.1.3 Properties

Each object has a set of propertiesthat are the data items of the object. VB allows access to all of the properties provided by each object, although the objects may restrict access to certain properties themselves. VB also provides a set of standard properties associated with each object, including the name of the object and its size and position if it is a graphical object. Examples of common properties are:

`BackColor`	MousePointer
`Caption`	Name
`DataField`	TabIndex
`DataSource`	Tag
`DragIcon`	Text
`Enabled`	Top
`Font`	Visible
`ForeColor`	WhatsThisHelpID
`Height`	Width
`Left`	

Most properties are set while designing the form and are used to indicate the initial and running state of the object. For example, the Name property can only be set while designing the form because it is read-only at run time. The position and color properties are seldom changed at run time, while the Enabled and Visible properties are commonly used to restrict user access to the control.

2.1.4 Methods

Each object has a set of methods; these are the functions that can be called to perform the operations that are allowed with the object. For example, a control that manages and presents a list of items would have an **AddItem** method to insert or append a new item. It would also have a **Clear** method to empty the list and render the display blank.

2.1.5 Events

Each object has a set of events; these are the notification messages that the object sends to its container (in this case, VB) when something happens with the object. VB provides these events as empty functions that you fill in if you want to respond to the notification. For example, a button control would have a **Click** event, which Visual Basic would present as an empty **Click** function that is called whenever the user presses the button by means of the mouse or keyboard. This function can even be called from VB code, just as a method would be called.

2.1.6 Naming Objects

Object names consist of several parts separated by periods. For example, to refer to a form, give the name of the form as follows:

```
MainForm
```

To refer to a control on the form, you would use the name of the form followed by the name of the control as in the following example:

```
    MainForm.CancelButton
```

To refer to a property of a form or control, you would use one of the preceding statements, followed by the property name, such as:

```
    MainForm.Name
    MainForm.CancelButton.Name
```

2.2 Understanding and Working with Projects

A VB project is the collection of files that make up your application. Visual NPL applications tie NPL programs to VB projects. When working with Visual Basic, you always have an open project, even when you first start up VB. The default project is named ***Project1*** and it has one blank form named ***Form1***.

2.2.1 Files That Make Up a Project

The main project file is a standard text file with an extension of **`.VBP`**. This file contains the basic project configuration information for your application, such as:

- The name of the project
- A list of the form and code files in the project
- A list of OCXs used by the project
- The name of the executable file produced when the project is compiled
- The size and position of the main VB window
- Miscellaneous configuration settings

Forms

There are two files used to represent a form. The main one has an extension of **`.FRM`** and the second, which is optional, has an extension of **`.FRX`**. The **`.FRM`** file is a text file containing:

- The values of any form properties that have been changed from the default values
- A list of the controls on the form
- For each control, the values of any properties that have been changed from the default values
- The source code for the events to which the form responds
- The source code for the subroutines contained within the form

The **`.FRX`** file stores graphics associated with the form, such as:

- The form's icon when minimized
- Any images appearing on the form
- Any other graphics used by the form

Both files are produced automatically whenever you create and change forms. There should be no need to edit these files manually.

Modules

Modules are source code files with an extension of **.BAS**. These are text files that contain the following:

- Variable declarations
- Constant declarations
- Type declarations
- Functions and procedures

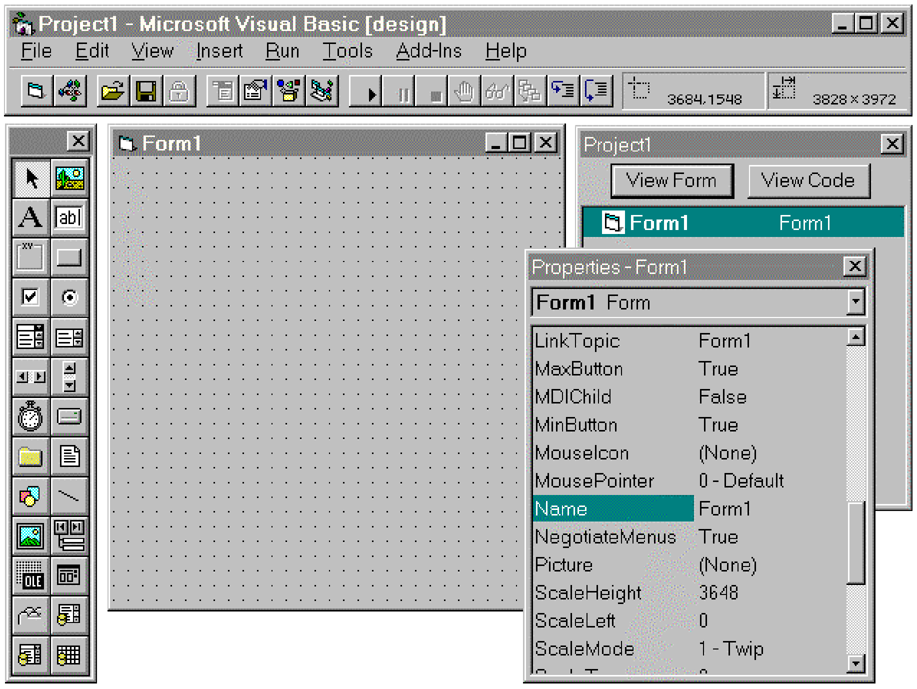
Modules are very similar to the individual modules within an NPL diskimage. As with NPL modules, you can declare things to be public (visible to the rest of your program) or private (can only be used within the module). However, unlike NPL, in Visual Basic you don't `INCLUDE` modules in other modules. Anything declared as public in one module is automatically usable in any other module. Furthermore, all VB modules are loaded into memory when your program starts running; there is no concept of overlays or dynamic loading of individual modules.

Classes

Classes are source code files with an extension of **.CLS**. These are text files that contain the definitions of any custom objects that you create within your application. The code for a class defines the properties and methods of the class. When a class has been defined, you can create and use objects of this type anywhere else in your program, just as you would use any of the standard objects available in VB. Your Visual NPL projects probably won't contain any classes, since there is no need for them.

2.2.2 And There Were Many Windows

When you open Visual Basic for the first time you will probably be amazed by the sheer number of windows that appear. Although it looks confusing, it is really quite simple. The following figure shows some samples of the windows that may be displayed.



LinkTopic	Form1
MaxButton	True
MDIChild	False
MinButton	True
MouseIcon	(None)
MousePointer	0 - Default
Name	Form1
NegotiateMenus	True
Picture	(None)
ScaleHeight	3648
ScaleLeft	0
ScaleMode	1 - Twip

Figure 2.1 Visual Basic Windows

There are seven different types of windows that can appear:

- The main VB window appears across the top of the screen. It contains the main menu and optionally a toolbar of shortcut icons.
- The project window contains the list of files in the project. This is where you access the forms and code of your project.
- The toolbox window contains icons for each control that can be added to a form.
- The properties window allows you to edit the properties of whatever form or control is currently selected (highlighted using the mouse).
- Multiple form windows are the graphical representations of the windows that appear in your application.

- Multiple code windows contain the code for your modules and classes.
- The debug window lets you look at data values while your program is running.

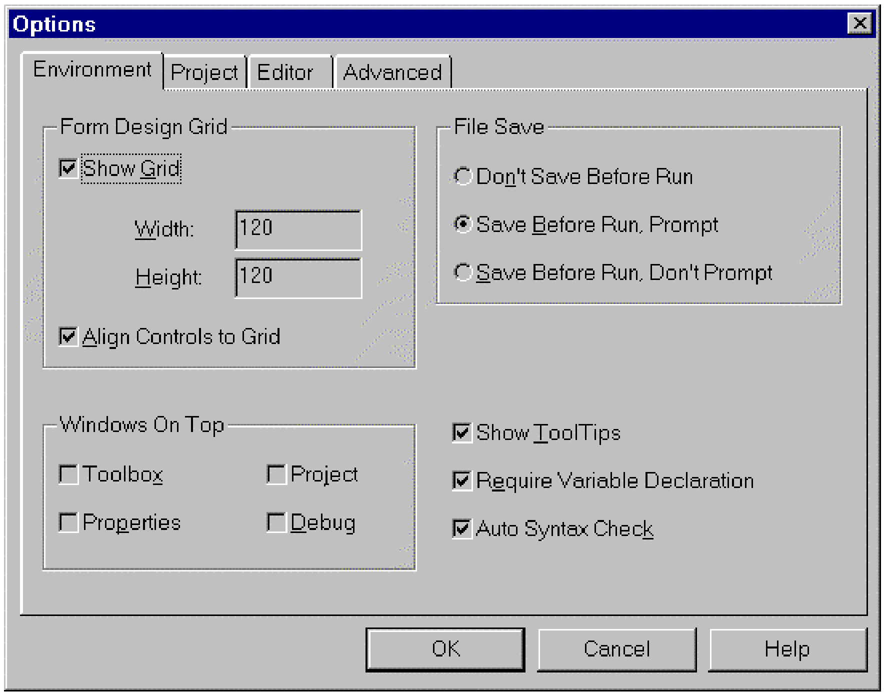
Because Visual Basic doesn't use a main window that contains all of the other windows, these windows can appear anywhere on your screen. All windows can be closed at any time and can be shown again using the *View* menu.

2.2.3 Useful Configuration Options

To change the configuration options of your VB environment and the current project, use the *Options* selection under the *Tools* menu. This presents you with the following option groups:

Environment	General options applicable to the VB environment
Project	Options specific to the current project
Editor	Options specific to the editing of code
Advanced	Miscellaneous advanced options specific to the current project

The following figure shows the *Environment* properties of the *Options* dialog box.



Options

Environment | Project | Editor | Advanced

Form Design Grid
[✓ Show Grid]
Width: 120
Height: 120
[✓ Align Controls to Grid]

File Save
Don't Save Before Run
[✓ Save Before Run, Prompt]
Save Before Run, Don't Prompt

Windows On Top
Toolbox
Project
Properties
Debug

[✓ Show ToolTips]
[✓ Require Variable Declaration]
[✓ Auto Syntax Check]

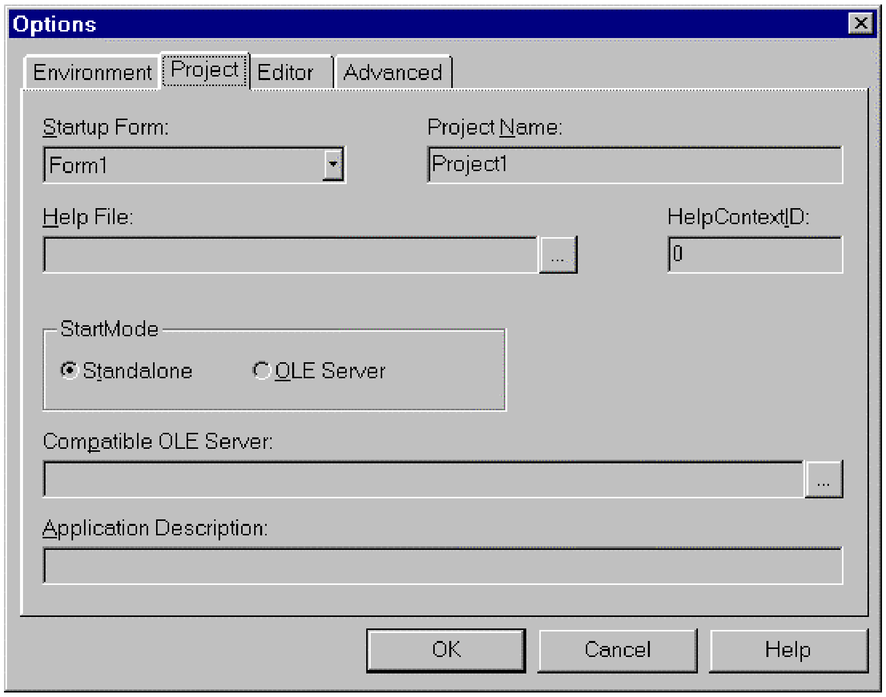
OK | Cancel | Help

Figure 2.2 ***Environment*** **Options**

The most frequently-used environment options allow you to do the following:

- Specify the grid used to align controls on a form. The default is to have both the *Show Grid* and *Align Controls to Grid* check boxes selected and the *Width* and *Height* both set to 120.
- Specify whether or not VB should save the project before running it. The safest option is to select *Save Before Run, Prompt*, which prompts you before running the project if any of the project files have been changed but have not been saved.
- Make sure that the *Require Variable Declaration* check box is selected so that no unforeseen assumptions are made about variable types within your code.

The following figure shows the *Project* properties of the *Options* dialog box.



Startup Form: Form1

Project Name: Project1

Help File:

HelpContextID: 0

StartMode: [✓ Standalone] OLE Server

Compatible OLE Server:

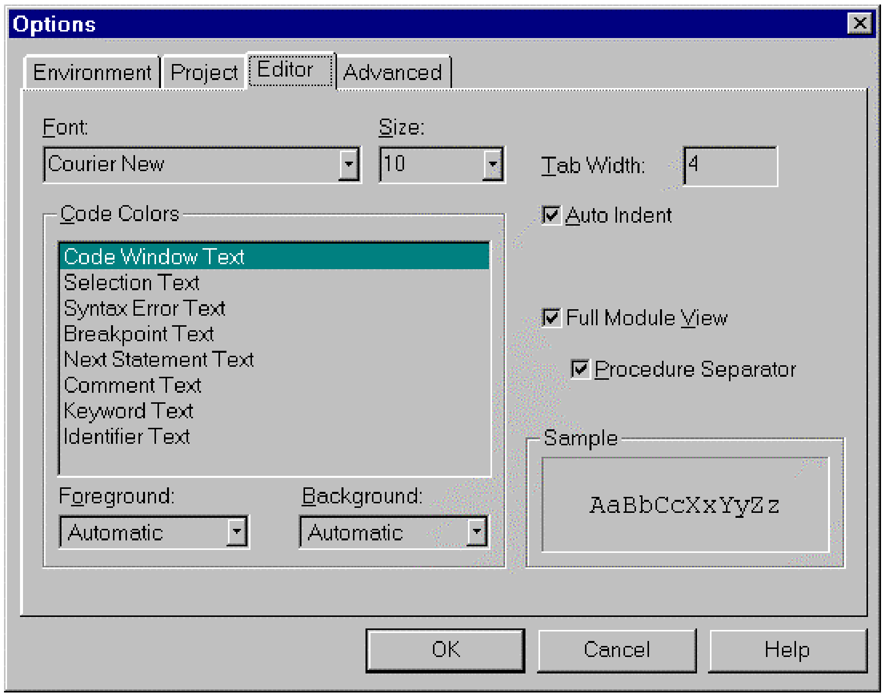
Application Description:

Figure 2.3 ***Project*** **Options**

The most important project options are the following:

- *Project Name*; this box should be changed to a unique project name,
- *Startup Form*; this box specifies the form at which your project will start executing. For Visual NPL programs, this should be set to *Sub Main*.

The following figure shows *Editor* properties of the *Options* dialog box.



Font: Courier New
Size: 10
Tab Width: 4
Code Colors
Code Window Text
Selection Text
Syntax Error Text
Breakpoint Text
Next Statement Text
Comment Text
Keyword Text
Identifier Text
[✓] Auto Indent
[✓] Full Module View
[✓] Procedure Separator
Foreground: Automatic
Background: Automatic
Sample
AaBbCcXxYyZz

Figure 2.4 ***Editor*** **Options**

Set the *Editor* options according to your personal preferences when editing code; you can change the following settings:

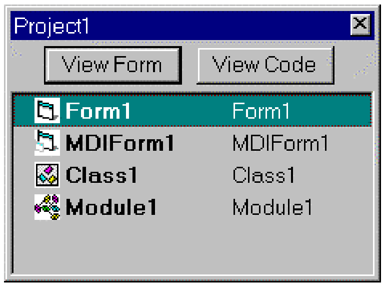
- The font used to show the code
- The color of specific syntax items so that you can highlight various language elements
- The width of a tab stop (the default is 4)
- The manner in which you view the code in a module, either as one long file or as a group of individual procedures and functions

The advanced options are set to default values; these should not be changed. For more information on these options, refer to the Visual Basic documentation.

2.2.4 Creating Forms, Modules, and Classes

To create a new form, module or class, use the appropriate menu command under the *Insert* menu. Notice that there are two types of forms listed on this menu, *Form* and *MDI Form*. An MDI (multiple document interface) form is the main window for an application with many subwindows, all of which are contained within the main window. For example, most word processors are MDI applications, where the main window has all the menus and toolbars and each open file is a simple subwindow within this main window.

The following figure shows a sample project window.



Form1	Form1
MDIForm1	MDIForm1
Class1	Class1
Module1	Module1

Figure 2.5 Project Window

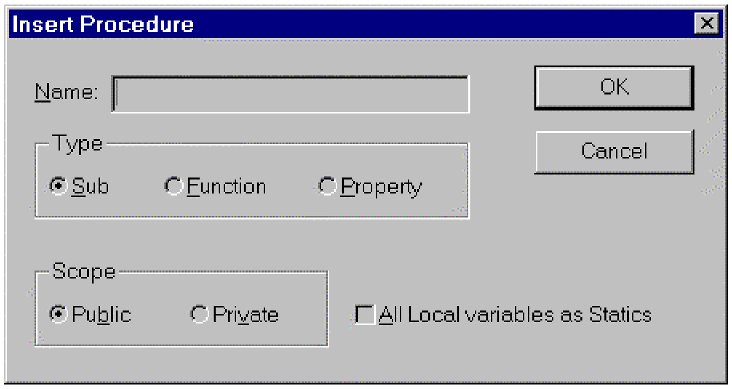
Regardless of which menu command you choose, a new file will be added to the project window and given a default name. Every file has two names: the disk-based file name (on the left) and the name used to refer to the file within your VB program (on the right). The first thing you should do is change both names to something meaningful. To change the file name, right-click the new entry and select the *Save File As* command. Fill in the new file name (without an extension) and click the *OK* button. To change the internal name, double-click the new entry in the *Project* window. In the *Properties* window, scroll down to the *Name* property and set it to what you want. That's it—you've got a new file and now you can do some real work with it!

2.2.5 Creating Procedures, Functions, and Properties

To create a new procedure, you must have a code editing window open and it must be selected. When this is done, you can select the *Procedure* command from the *Insert* menu. This presents you with a window where you can specify:

- The name of the new item
- Whether the item is a procedure, a function, or a property
- Whether the item is private to the file it is in or is publicly available to all files

The following figure shows an example of the *Insert Procedure* window.



Type: [✓ Sub] Function Property

Scope: [✓ Public] Private

☐ All Local variables as Statics

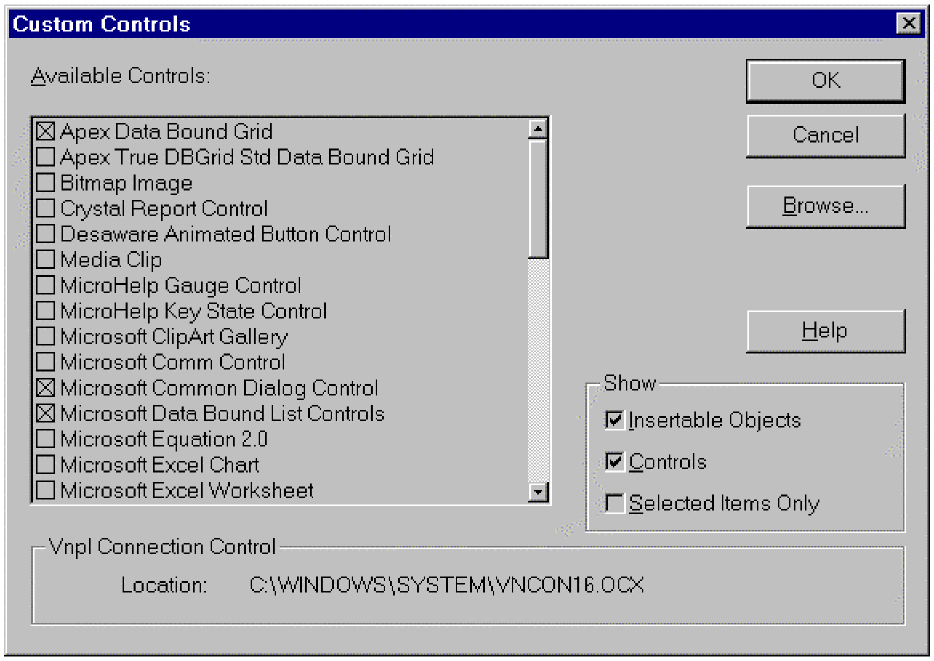
Figure 2.6 ***Insert Procedure*** **Window**

When you click the *OK* button, the appropriate code is created in the current file and the cursor is positioned within the new subroutine. You can then add any parameters you want and fill in the code.

2.2.6 Using Third-Party Controls

To use a third-party control (OCX) you must first add the control to your project. You do this using the *Custom Controls* command under the *Tools* menu. Visual Basic presents you with a list of all of the controls available on your system. Each control has a check box beside it, indicating whether or not the control is included in the current project.

The following figure is an example of the *Custom Controls* dialog box.



Custom Controls

Available Controls:

- ☒ Apex Data Bound Grid
- ☐ Apex True DBGrid Std Data Bound Grid
- ☐ Bitmap Image
- ☐ Crystal Report Control
- ☐ Desaware Animated Button Control
- ☐ Media Clip
- ☐ MicroHelp Gauge Control
- ☐ MicroHelp Key State Control
- ☐ Microsoft ClipArt Gallery
- ☐ Microsoft Comm Control
- ☒ Microsoft Common Dialog Control
- ☒ Microsoft Data Bound List Controls
- ☐ Microsoft Equation 2.0
- ☐ Microsoft Excel Chart
- ☐ Microsoft Excel Worksheet

Show

- ☑ Insertable Objects
- ☑ Controls
- ☐ Selected Items Only

Vnpl Connection Control

Location: C:\WINDOWS\SYSTEM\VNCON16.OCX

Figure 2.7 ***Custom Controls*** **Window**

To indicate the functionality you want, select the check boxes you want to add and clear the check boxes that you don’t want. When you click the *OK* button, the toolbox window is redrawn to include an icon for each control now in the project. As a safeguard, VB doesn't let you clear any control that is currently being used on one of your forms.

2.2.7 Making an EXE File

To compile your project into an **.EXE** file, use the *Make EXE* command on the *File* menu.

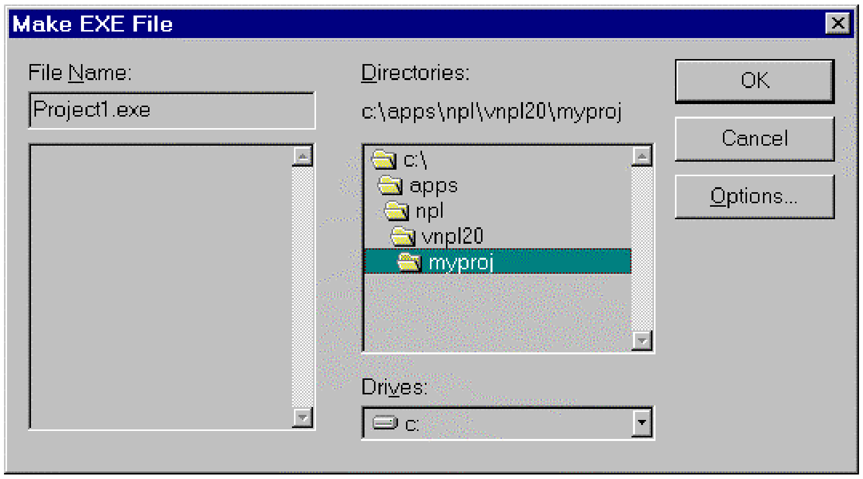


Figure 2.8 Make EXE Window

With this window you can set the name of the executable file. By pressing the *Options* button, you set the icon used by the application when minimized and the version information stored in the **.EXE**.

The following figure is an example of the *EXE Options* dialog box.

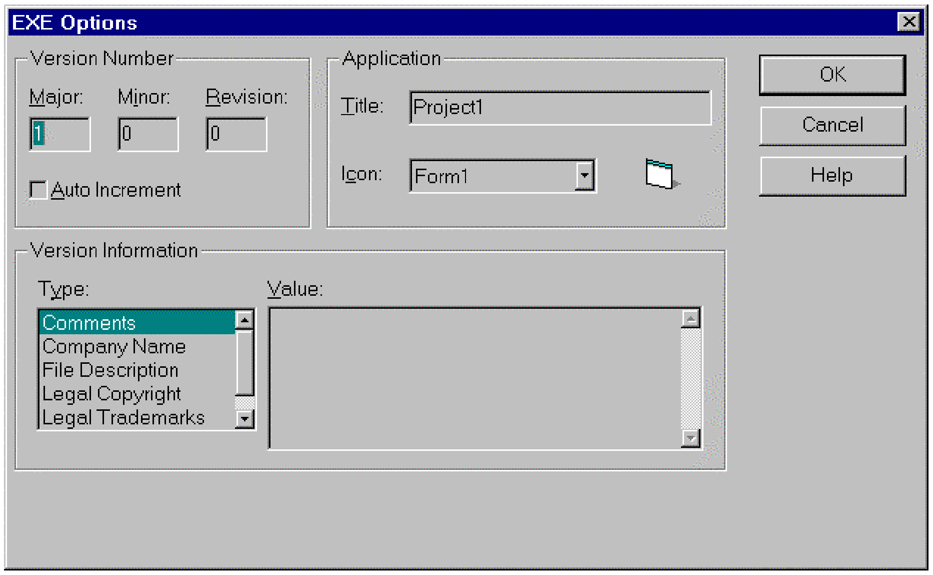


Figure 2.9 ***EXE Options*** **Window**

When you click the *OK* button in the first window, VB compiles your program and produces the **.EXE** file. If you try to overwrite an existing **.EXE** file, you will be warned and given the chance to stop the process before any damage is done. It is not necessary to create an **.EXE** file in order to test your VB application. You can then run the program from within the development environment at any time.

2.2.8 Running and Debugging Your Program

To run a program, use the *Start* command under the *Run* menu. This compiles the program, saves it to the disk (depending on your *Options* settings), and runs the program. To pause the program while it is running, use the *Break* command under the *Run* menu. To continue a paused program, use the *Continue* command under the *Run* menu. To stop a program completely, use the *End* command under the *Run* menu.

You can also run, pause, and continue a program by pressing the F5 key. If the program is not running, then pressing F5 starts or continues it. If the program is running, then pressing F5 pauses it. While a program is paused, you can use the *Debug* window to examine or change your program's variables.

To set a breakpoint in your program, at the line where you want the program to break, press F9. To remove the breakpoint, press F9 again. You can only do this before running a program or when the program is paused. When a breakpoint is in place, a running program will pause whenever it is about to execute that line.

2.3 Designing Forms

Designing a formis the process of graphically creating and modifying everything to do with the appearance of the form. This process involves the following:

- Modifying the form’s appearance
- Creating controls and modifying their appearance
- Creating a menu for the form

2.3.1 Selecting Objects with the Mouse

To modify the properties for an object, you must first select the object with the mouse. To selecta particular control, click the mouse anywhere on the control. Notice that the control is marked with small black squares at its corners and halfway along each side.

The following two-part figure shows examples of what is displayed when a control is selected.

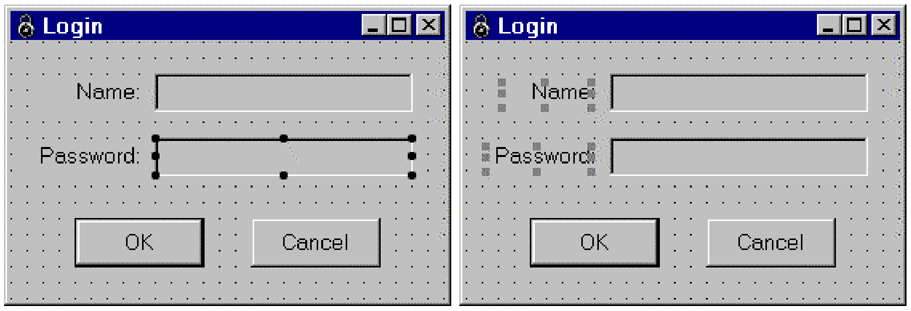


Figure 2.10 Control Selection

To select more than one control, press and hold down the CTRL key and click the mouse on each control that you want to select. Each control becomes marked with small gray squares at its corners and halfway along each side.

You can also click the form background and drag the mouse around the controls you want to select. To select the form itself, click the mouse anywhere on the background of the form. Any controls that were selected will become cease to be selected and all of the small black or gray marks will disappear.

The following figure shows an example of what is displayed when a form is selected.

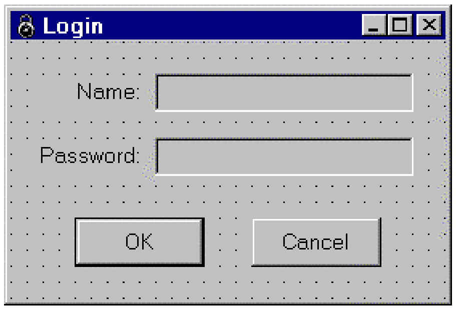
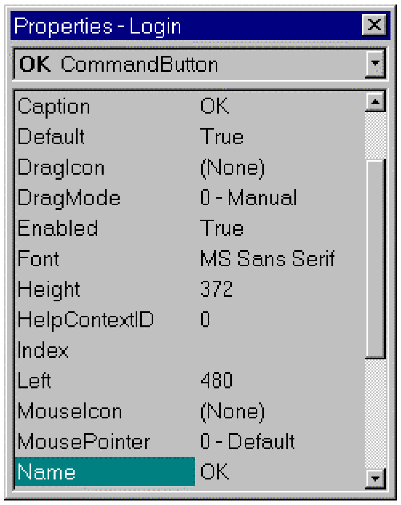


Figure 2.11 Form Selection

Each time a new object is selected, the *Properties* window is redrawn with the properties of the new object. If there is more than one object selected, the *Properties* window is redrawn with the properties that are common to all of the selected objects.

2.3.2 Setting Form and Control Properties

The property window contains a drop-down list of the form and control names at the top of the window. This list shows the name of the currently selected object; it shows no name if there is more than one object selected. You can use this list to select the form or a single control. The properties list is redrawn with the properties of the selected object.


Properties - Login

OK CommandButton

Caption	OK
Default	True
DragIcon	(None)
DragMode	0 - Manual
Enabled	True
Font	MS Sans Serif
Height	372
HelpContextID	0
Index	
Left	480
MouseIcon	(None)
MousePointer	0 - Default
Name	OK

Figure 2.12 Properties Window

Below the list of form and control names is the list of properties for the currently selected object. The left side of the list gives the property name and the right side gives the property's value. To set a particular property, scroll down the list and click the value that you want to change. If it's an editable field, depending on the type of property, you may be able to update the value in place using one of the following methods:

- Enter a text value.
- Select an item from a drop-down list.
- Present a window to select or edit the item.

Although there are hundreds of form and control properties that can be changed, almost all of them default to reasonable values. Most of the time, you will only need to change two or three properties for each object.

2.3.3 Adding and Deleting Form Controls

To add a new control, double-click the control's icon in the *Toolbox* window. This creates the new control with a default size in the middle of the form. To manually define the initial size and position of the control, click the icon in the *Toolbox* window and then draw the control on the form. Do this by clicking the form's background and dragging the mouse to draw a rectangle where you want the control to appear. Release the mouse button to cause the control to be created. When a new control is created, it is given default values for all of its properties and it is selected as the current control. The *Properties* window is updated accordingly.

The following figure shows an example of how to add controls to a form.

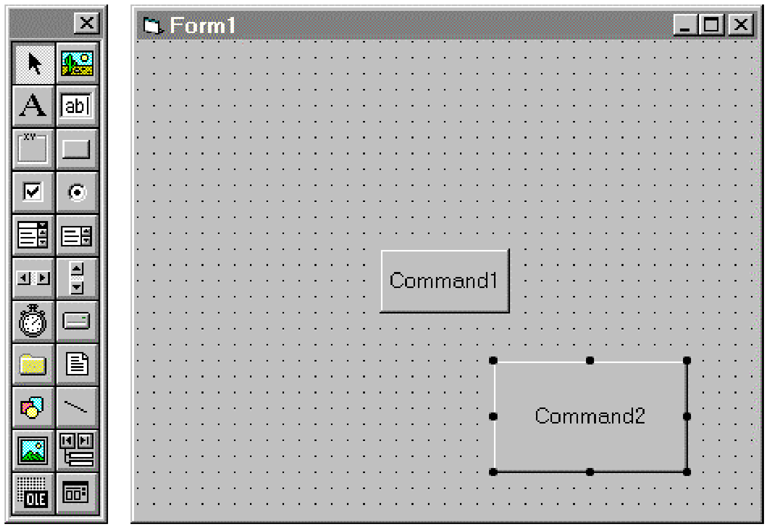


Figure 2.13 Adding and Deleting Controls

To delete a control, do one of the following:

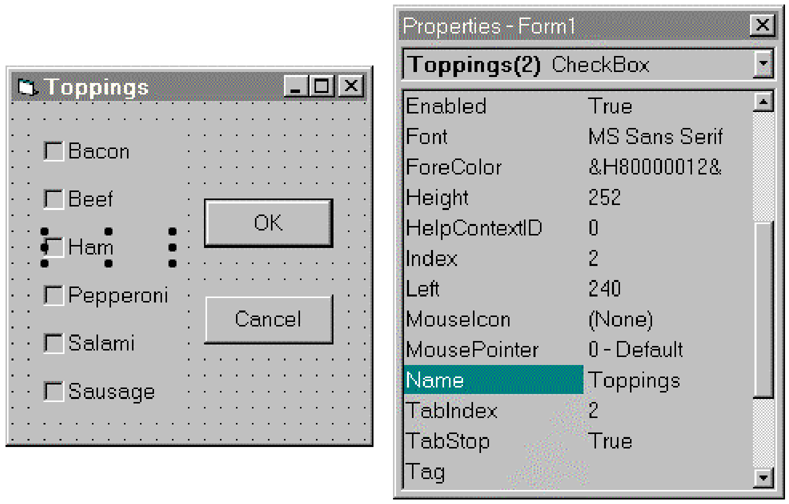
- Select the control and press the DEL key.
- With the control selected, select the *Delete* command in the *Edit* menu.
- Right-click the control and select the *Delete* item from the menu that appears.

When you delete a control, Visual Basic does not delete the code that you have created for the events of the control. You must delete this code manually using the code-editing window.

2.3.4 Form Control Arrays

Typically, every control on a form has a unique name. However, sometimes it is desirable to group several controls together for ease of programming. For example, when using a series of check boxes or option buttons that all apply to the same thing, it is easier to use a loop to go through an array than to test for some condition on a whole group of unique names.

The following two-part figure shows examples of how to define control arrays.



Properties - Form1

Toppings(2) CheckBox

Enabled	True
Font	MS Sans Serif
ForeColor	&H80000012&
Height	252
HelpContextID	0
Index	2
Left	240
MouseIcon	(None)
MousePointer	0 - Default
Name	Toppings
TabIndex	2
TabStop	True
Tag	

Figure 2.14 Control Arrays

To create a control array choose the first of the controls that you want to be grouped. Set its Name property to the name of the array and set its Index property to 0. If the control is not already a member of an array, then its Index property is blank. Now change the other control's Name properties to the name of the array. Visual Basic automatically chooses the next Index value in the array for each control, so make sure to define the values in the order that you want them to appear in the array. Whenever you refer to one of these controls, you have to specify the index number as an array index following the control name, as in the following:

```
Toppings(2).Value
```

Consider the following information when working with control arrays:

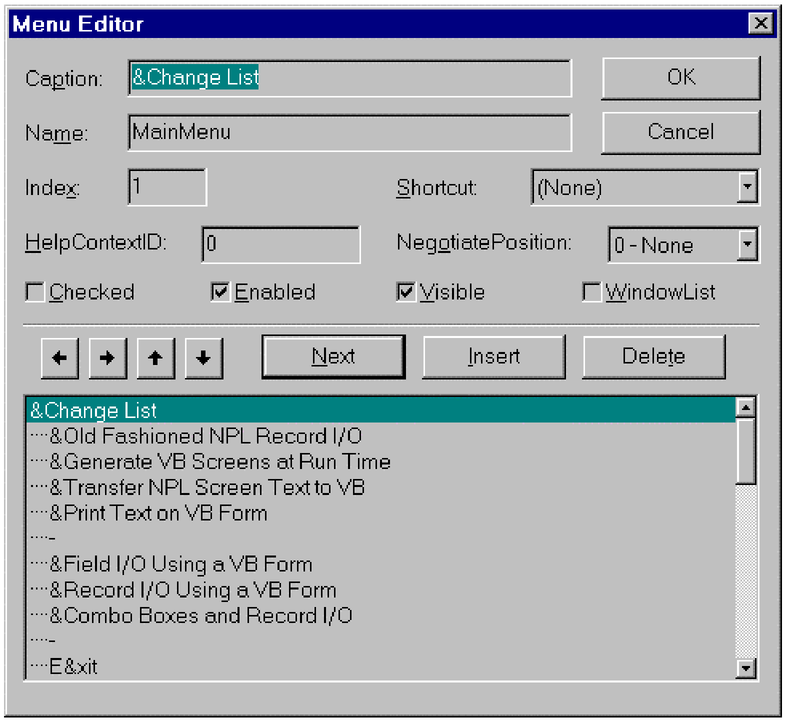
- All of the controls in the array must be of the same type.
- The index numbers can start at any number you want.
- The index numbers don’t have to be consecutive.
- All events for the array are passed an Index as the first parameter to identify which control is triggering the event (that is, the controls in the array share the same event code).

2.3.5 Setting the Tab Order

One of the most important things to do when creating a form is to set the tab order for the controls. This is the order in which the controls will be traversed when the user presses the TAB key. To set the tab order for your controls, set the TabIndex property for each control. When the TAB key is pressed, the cursor moves among the controls in ascending order by the TabIndex property. This should be one of the last things that you do with a form, because controls often get moved around quite a bit before the final form design is determined and you may waste time and effort if you try to set the tab order before the design is complete.

2.3.6 Creating Menus

To create a menu for a form, select the *Menu Editor* command on the *Tools* menu. This will bring up the menu editor window.


Menu Editor

Caption: &Change List

Name: MainMenu

Index: 1

Shortcut: (None)

HelpContextID: 0

NegotiatePosition: 0 - None

☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&Change List
····&Old Fashioned NPL Record I/O
····&Generate VB Screens at Run Time
····&Transfer NPL Screen Text to VB
····&Print Text on VB Form
····-
····&Field I/O Using a VB Form
····&Record I/O Using a VB Form
····&Combo Boxes and Record I/O
····-
····E&xit
```

Figure 2.15 ***Menu Editor*** **Window**

When you are creating a menu, you are creating several control arrays, one for each group of items at the same level on the same menu. The *Name* and *Index* fields in the *Menu Editor* window are the same as the *Name* and *Index* properties in the *Properties* window. To edit the menu, do the following:

- Use the *Insert* button to add a new menu command above the currently selected item.
- Use the *Delete* button to remove the currently selected item.
- Use the arrow buttons to move the items around within the menu and to change the nesting level of menu commands.
- Use the *Enabled* and *Visible* check boxes to control access to the menu commands.
- Use the *Checked* check box to put check marks beside menu commands.
- Use the *WindowList* check box with an MDIform for the top-level menu that holds the list of open windows.
- Use the *Shortcut* drop-down list to select shortcut keystrokes for menu commands.
- Set the caption to a hyphen (-) when you want to create a separator bar on a menu.
- Select the underlined letter in each menu command by putting an ampersand (&) in front of it.

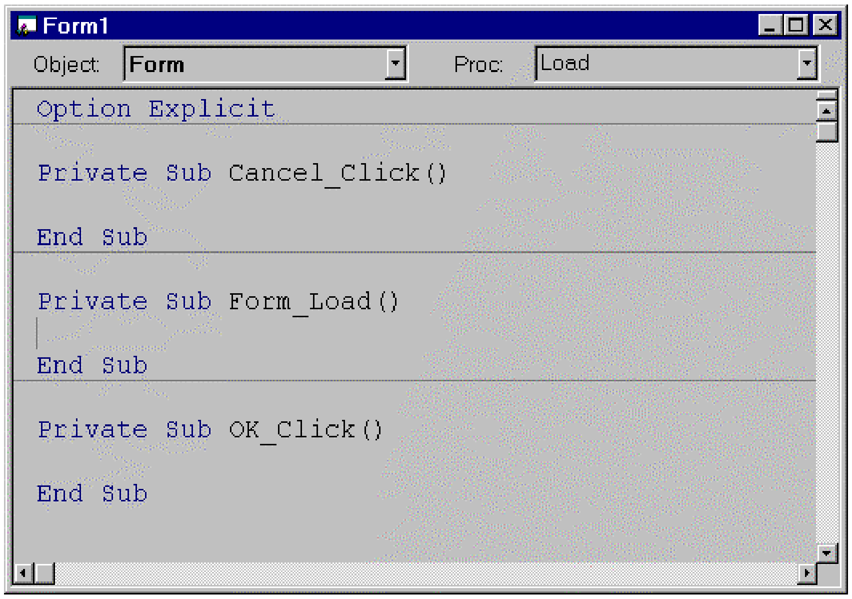
2.4 Editing Code

To open the code-editing window for a module or a class, double-click the appropriate line in the *Project* window. To open the code-editing window for a form, do one of the following:

- Double-click the appropriate line in the *Project* window, and then double-click the form background or one of the controls.
- Click the appropriate line in the *Project* window, and then click the *View Code* button in the *Project* window.

The *Code Editor* window contains two drop-down lists at the top of the window, one for selecting the object being edited (the *Object* list) and one for selecting the procedure within the object (the *Proc* list). The rest of the window is devoted to the actual code editor.

The following figure shows an example of the *Code Editor* dialog box.



```
Option Explicit

Private Sub Cancel_Click()

End Sub

Private Sub Form_Load()

End Sub

Private Sub OK_Click()

End Sub
```

Figure 2.16 ***Code Editor*** **Window**

When editing modules:

- The *Object* list only contains an entry named *(General)*.
- The *Proc* list for the *(General)* object contains *(declarations)* and the names of all of the procedures in the module.

When editing classes:

- The *Object* list contains one entry named *(General)* and one entry named *Class*.
- The *Proc* list for the *(General)* object contains *(declarations)* and the names of all of the local procedures in the class.
- The *Proc* list for the *Class* object contains the **Initialize** and **Terminate** methods, the names of any methods created by the developer, and the names of any property procedures created by the developer.

When editing forms:

- The *Object* list contains one entry named *(General)*, one named *Form*, and the names of each control (including menus) on the form.

- The *Proc* list for the *(General)* object contains *(declarations)* and the names of all of the local procedures in the form.
- The *Proc* list for forms and controls contains the names of all of the events for the object.

When you first create a new form or control, there is no code for any of the events. To add code for a particular event:

- Select the appropriate item from the *Object* list and then click the event name in the *Proc* list.
- If the event doesn't exist, it will be created and the cursor will be positioned within it.
- If the event already exists, the cursor will be moved to the start of the event procedure.

The view in the editor portion of the window will depend on the settings of the *Full Module View* and *Procedure Separator* check boxes in the *Editor* tab of the *Options* window. If *Full Module View* is not checked, then you will only see the code for procedure that is currently selected in the *Proc* list. If it is checked, then all of the procedures in all of the objects will appear as one long file. In this case, if *Procedure Separator* is checked, then there will be a separator line between each procedure.

C H A P T E R 3

Visual NPL Fundamentals

This chapter is an introduction to Visual NPL programming. It discusses:

- The overall structure of Visual NPL and how it works
- Event-driven programming
- Accessing VB objects, properties, and methods from NPL
- Handling events by calling NPL functions from VB
- Managing the NPL run-time’s main window
- Miscellaneous programming details

3.1 How It All Works

The following figure shows the interactions of various programming elements in Visual NPL.

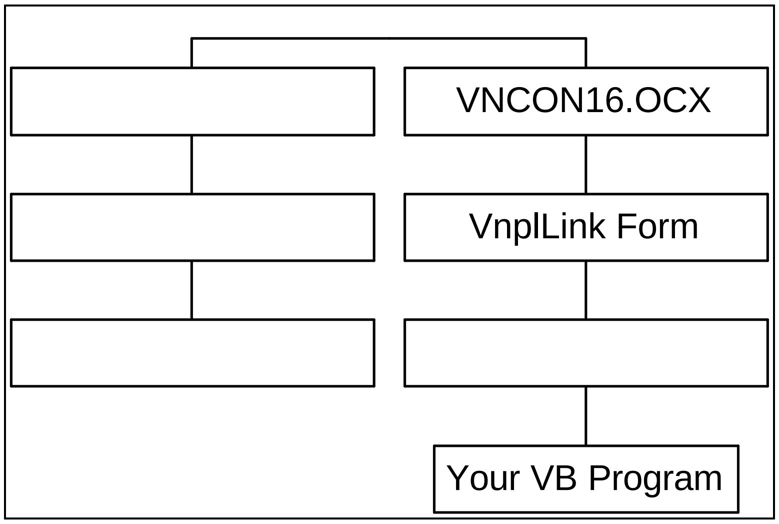


Figure 3-1 Visual NPL Structure

Visual NPL works by creating a link between an NPL program and a VB program. This communications link is created by using an NPL external library (**`VNPL16.DLL`**) and a VB control (**`VNCON16.OCX`**) to send messages back and forth between the two environments.

In NPL, access to the link is provided by a set of subroutines in modules `"Vnpl"` and `"VnplDev"` in diskimage **`VNPL.NPL`**. In VB, access to the link is provided by subroutines in modules **VnplUtil** and **VnplDev** in files named **`VNPLUTIL.BAS`** and **`VNPLDEV.BAS`**, respectively. Also, there is a VB form named **VnplLink** in the file named **`VNPLLINK.FRM`**, which is where the connection control actually exists.

3.1.1 NPL Diskimage

The `"Vnpl"` and `"VnplDev"` modules are in the **`VNPL.NPL`** diskimage. The `"Vnpl"` module is a protected module that contains most of the interface provided by Visual NPL. The `"VnplDev"` module contains only the constants, variables and routines that you are allowed to change, which include:

- Device number of the diskimage containing `"Vnpl"`
- Delimiter used to separate a list of parameters
- Minimum size of a `/POINTER` string parameter passed from VB to NPL
- Maximum number of forms or controls that can be returned in a list
- Constants to convert NPL screen positions to VB window positions
- Error-handling routines and textual error messages
- Textual color names
- Keyboard translation strings

⇔ **To use Visual NPL from an NPL program**

Set the device number (`_VnSys`) of the diskimage containing `"Vnpl"`. Then, every NPL module that uses Visual NPL should begin with the following two `INCLUDE` statements:

```
INCLUDE T "VnplDev"
INCLUDE T#_VnSys,"Vnpl"
```

For purposes of these statements, `"VnplDev"` *must be in Device #0* and `"Vnpl"` can be in any device. Typically, you will copy the `"VnplDev"` module from **`VNPL.NPL`** into your main diskimage, which should be Device #0. Then you will set the `_VnSys` constant to the device number of the entry that contains the path for **`VNPL.NPL`**. The advantages of this approach are:

- You only need one copy of **`VNPL.NPL`** on your computer, making it easier to update to future versions of Visual NPL.
- You can make project-specific changes to `"VnplDev"` without affecting the original `"VnplDev"`.
- The **`VNPL.NPL`** diskimage can be moved to a different device number by changing the value of `_VnSys` in `"VnplDev"`.

3.1.2 VB Modules and Linkage Form

The VB **VnplUtil** and **VnplDev** modules are in files named **`VNPLUTIL.BAS`** and **`VNPLDEV.BAS`**, respectively and the **VnplLink** form is in the file named

`VNPLLINK.FRM`. The **VnplLink** form contains a connection control that is used by the routines in **VnplUtil** and **VnplDev**. The **VnplUtil** module contains most of the interface provided by Visual NPL.

Note The **VnplUtil** module should not be changed, although Visual Basic will allow you to do so.

The **VnplDev** module contains only the constants and routines that you are allowed to change. This includes the following:

- Delimiter used to separate a list of parameters
- Constants to convert NPL screen positions to VB window positions
- Function used to register your forms
- Function used to register your controls created "on the fly"
- Function used to add commands to the NPL `VnCmd` function
- Visual Basic **Main** procedure

⇔ **To use Visual NPL from a VB project**

1. Enable the Visual NPL control by selecting the *Custom Controls* command under the *Tools* menu and selecting the *Vnpl Connection Control* check box.
2. Add the **VnplLink** form to your project.
3. Add the **VnplUtil** and **VnplDev** modules to your project.
4. Set up the VB project so that it starts at the **Main** procedure in **VnplDev**.

If you are adding Visual NPL to an existing VB project that already has a **Main** procedure, merge the two into one. Make sure to put the code in **VnplDev** at the very start of the new **Main** procedure.

3.1.3 Building the Connection

To build the connection between NPL and VB, each program must call a routine to initialize communications. In the VB program this code is in the **Main** procedure in **VnplDev**. As shown in the following sample, the code calls the **Init** method of the **VnCon** control on the **VnplLink** form. The method is passed the name of the VB executable file, which is used as a key when trying to establish communications with NPL:

```
VnplLink.VnCon.Init(App.EXEName)
```

After this has been passed, the connection control is waiting for connection by an NPL program that can detect the base name of the executable file (the file name without the full path and without the **.EXE** file-name extension). To complete the link from the NPL program, you must call the NPL `'VnOpen` procedure with the base name as follows:

```
'VnOpen("BASENAME")
```

This procedure looks for a connection control with the specified base name that is waiting for a connection. If it can't find one, it looks for the executable file on the standard Windows search path and runs it. It then looks again for the connection control. If it still can't find it, or if it couldn't find or run the executable, the procedure generates a Visual NPL error. Otherwise, the connection is made and your program continues.

This approach allows you to run your Visual NPL program using either the VB development environment or the executable file produced by VB. To run the program with the development environment, you must run the VB program first, and then run the NPL program. To run the program with the VB executable, you only need to run the NPL program.

When your program is finished, it must close the connection that it built. This is done by calling the `'VnClose` procedure from your NPL program as follows:

```
'VnClose
```

This tells the VB program to end and close the connection. You don't have to do anything in VB.

3.1.4 Multiple Connections

Although there generally isn't much need to do so, it is possible for a single NPL program to connect to up to 32 VB programs. Each time you successfully call `'VnOpen`, it creates a unique VB application number identifying the new connection. The `'VnOpen` routine also sets an internal variable to this number, which is then used by all subsequent calls to the Visual NPL routines rather than having to pass it as a parameter every time. This number is referred to as the *current application number*. This makes it easy to connect to a single

VB application because you don’t have to do anything with application numbers at all.

On the other hand, connecting to more than one VB program requires some additional steps:

- Retrieve the current application number after each call to `'VnOpen` and record it somewhere for later use.
- Make sure that the current application number is set correctly for every call to a Visual NPL routine.
- Close all connections when done.

⇔ **To obtain the current application number**

- Call the `'VnGetAppNum` function immediately after calling `'VnOpen` as follows:

```
'VnOpen("BASENAME")
AppNum='VnGetAppNum
```

⇔ **To set the current application number**

- Call the `'VnSetAppNum` procedure before each group of calls to the same VB program as follows:

```
'VnSetAppNum(AppNum)
```

⇔ **To close all open connections**

- Call the `'VnCloseAll` procedure when your program is shutting down as follows:

```
'VnCloseAll
```

3.2 Event-Driven Programming

VB is an *event-driven progamming* environment. This means that all processing is done in response to a user action, such as selecting a menu command, typing a character, or clicking a button (called *events*). Every time an event occurs, Visual Basic calls a predefined procedure (for which you fill in the code). For most events you will handle, just call an NPL `PROCEDURE` using VB procedure **VnCallProc** as follows:

```
Private Sub NextButton_ Click()
  Rem VB
  VnCallProc "GotoNextRecord"
End Sub


PROCEDURE 'GotoNextRecord/PUBLIC
  ;NPL
END PROCEDURE 'GotoNextRecord
```

3.2.1 Mainline Program

The mainline of a Visual NPL program is usually very simple and includes:

- Initialize your application.
- Open the connection with VB.
- Start up the main form and wait for it to close.
- Close the connection with VB.
- Terminate your application.

In its simplest form, an NPL example would contain the following:

```
;open the link to the VB app
'VnOpen("BASENAME")
;
;hide the run-time window
'VnSetNplWndShow(_VnHide)
;
;load, process and unload the main form
'DoMainForm
;
;Close the link to the VB app
'VnClose
;
;Optionally show the run-time window again
```

```
'VnSetNplWndShow(_VnShow)
;
$END
```

where `'DoMainForm` is a procedure that handles the creation, processing, and shutdown of the main form in your application. There is nothing special about this procedure; it's just the mainline of a form.

3.2.2 Handling a Form

Each form in your program should have a *form handler*, a procedure that serves as the form's mainline. It displays the form and puts NPL into a suspended state. Eventually some event, such as clicking a *Close* button, causes the mainline to wake up. It then closes the form:

```
PROCEDURE 'DoMainForm
;NPL
;
;Show the form modelessly
'VnMethod("MainForm.Show"," ")
;
;Wait for 'VnWakeup to be called
'VnSleep
;
;Unload the form
'VnCmd("MainForm","Unload"," ")
;
END PROCEDURE 'DoMainForm
```

The only VB event procedure that has to be filled in is for the **Click** event of the *Close* button:

```
Private Sub CloseButton_ Click()
  Rem VB
  VnCallProc "VnWakeup "
End Sub
```

The `'VnWakeup` procedure causes the `'VnSleep` call in the mainline to return. In other words, it wakes up the mainline, which then continues executing normally.

Warning When calling `'VnSleep`, there *must always be* a call to `'VnWakeup` to cause it to return.

3.3 Using Objects

Most Visual NPL programming is centered around manipulating VB objects, such as forms and controls. There are two different ways in which objects can be accessed:

- By specifying the full name of the object in all calls that use the object
- By getting a reference to the object, using the reference in all calls that use the object, and then releasing the reference when done.

Internally, each time you refer to an object by name, the name is resolved into an object reference. This means that using object references is going to be faster than using object names. However, in many cases, you only need to use a particular object in one or two statements in a procedure. In this case, the overhead of getting and freeing the object reference will negate the speed advantage. In general, only use object references for objects that you will be working with extensively.

3.3.1 Object Names

Object names consist of several parts separated by periods. For example, to refer to a form you would simply give the name of the form:

```
MainForm
```

To refer to a control on the form, you would use the name of the form followed by the name of the control:

```
MainForm.CancelButton
```

To refer to a property of a form or control, you would use one of the preceding followed by the property name:

```
MainForm.Name
MainForm.CancelButton.Name
```

These names are just strings in the NPL program. For example, the following code will print the names of the main form and the *Cancel* button:

```
DIM Name$100
;NPL
;
PRINT 'VnGetAlf$("MainForm.Name")
;
Name$="MainForm.CancelButton.Name"
PRINT 'VnGetAlf$(Name$)
```

It will sometimes be necessary to build an object name from individual parts that are stored in separate variables. For example, you could pass the name of an object to a procedure that prints the name of the object. In this case you would have to append `.Name` to the form name. The `'VnObj$` function can be used for exactly this purpose. It accepts two parameters and returns the combined values separated by a period.

```
PROCEDURE 'PrintObjName(/POINTER ObjName$)
;NPL
;
PRINT 'VnGetAlf$('VnObj$(ObjName$,"Name"))
;
END PROCEDURE 'PrintObjName
```

3.3.2 Object References

Object references are created by calling the `'VnGetObj$` function with the name of an object. The function returns a string containing the reference to the object. This reference is a value (a pointer in C programming) with a prefix and suffix that are used to differentiate the reference from an actual object name. The reference will be `_VnObjLen` characters long.

```
DIM Obj$_VnObjLen
;NPL
;
Obj$='VnGetObj$("MainForm")
;
PRINT 'VnGetAlf$('VnObj$(Obj$,"Name"))
;
'VnFreeObj(Obj$)
```

It is imperative that for each call to `'VnGetObj$`, there is a corresponding call to `'VnFreeObj`. This releases the internal Windows resources used to create references. Without this call, your program will be creating a Windows memory leak, which will eventually cause the system to stop.

With object references, it almost always necessary to call `'VnObj$` as in the preceding section, because most functions expect an object followed by a property or method name.

3.3.3 Registering Your Forms

Internally, Visual NPL resolves all object names into object references. In order to do this it needs to know about the forms that you have created in your VB program. Unfortunately, VB doesn't provide this information, so you must specify it. For each form that you create, you must add a `Case` to the `Select` statement in the **VnSetObj** function in the ***VnplDev*** module in your VB program. The text of the `Case` should be the name of the form and the single line of code for the `Case` should set the `Obj` parameter to the form itself:

```
Case "MainForm"
Set Obj = MainForm
```

This allows the form to be used in the NPL program, for example:

```
PRINT 'VnGetAlf$("MainForm.Caption")
```

3.3.4 System Objects

Along with the forms that you create, there are several system objects that are available to your NPL program.

System Object	Description
`App`	Application
`Clipboard`	Clipboard
`Err`	Last general error
`Error`	Last data access error
`Forms`	Currently loaded forms
`Printer`	Current printer
`Printers`	List of available printers
`Screen`	Screen

Table 3-1 System Objects

These objects can be used anywhere that any other object can be used, for example:

```
PRINT 'VnGetAlf$("App.EXEName")
PRINT 'VnGetNum("Printer.Copies")
```

3.4 Accessing VB from NPL

Visual NPL gives youcomplete control over the objects in your Visual Basic program. You can access properties, call methods, retrieve collections, perform specialized commands, and even create your own commands.

3.4.1 Setting and Getting Properties

The most fundamental thing you'll want to do from NPL is to set and get the various properties of an object. There are several types of properties, all of which can be setwith one of these procedures:

`'VnSetAlf`	Sets a property value from a string
`'VnSetNum`	Sets a property value from a number
`'VnSetObj`	Sets a property value to an object

These procedures all take two parameters: the first is the property to be set and the second is the value to which the property should be set. For example:

```
'VnSetAlf("MainForm.Caption","Title")
'VnSetNum("MainForm.Visible",_True)
'VnSetObj("MainForm.CheckBox.Container","MainForm")
```

Corresponding to the Set procedures are the three Get functions:

`'VnGetAlf$`	Gets a property value as a string
`'VnGetNum`	Gets a property value as a number
`'VnGetObj$`	Gets a property value as an object reference

These functions all take one parameter, the property to get. For example:

```
PRINT 'VnGetAlf$("MainForm.Caption")
PRINT 'VnGetNum("MainForm.Visible")
HEXPRINT 'VnGetObj$("MainForm")
```

In order to use these functions you must know the type of the property that you want to access. Even so, the Set procedures will try to convert their parameter into the appropriate type for the property. Similarly, the Get functions will try to convert the property value into the type that they return. For example, `'VnGetAlf$` will convert a numeric property into a string and return the string version of the number.

3.4.2 Calling Methods

In addition to accessing properties, you will often need to call an object's methods in order to perform object-specific operations. The 'VnMethod procedure takes two parameters, the method to be called and a string containing the parameters to be passed to the method. The individual parameters are separated by the global delimiter character _VnDelim$, the default for which is the "pipe" symbol (|).

```
'VnMethod("MainForm.ListBox.AddItem","Item")
'VnMethod("MainFormListBox.Clear"," ")
'VnMethod("MainForm.Move","100|100")
```

Each parameter will be converted to the appropriate type before being passed to the method.

3.4.3 Getting Collections

A collection is a VB object that contains an array of other VB objects. There are several collections that you may want to use while working with VB programs and Visual NPL provides procedures for accessing the most common information in these collections:

Procedure	Collection
`'VnGetFormCtrlList`	Controls on a form
`'VnGetLoadedFormList`	Currently loaded forms
`'VnGetPrinterList`	Available printers
`'VnGetPropInfoList`	Properties of an object

Table 3-2 Available Collections

In all cases, the procedures return the collection information in two global variables:

- A numeric indicating the number of elements in the collection
- A string array with the information for each object in the collection

For example:

```
DIM I
;
'VnGetLoadedFormList
;
PRINT "Loaded forms:"
FOR I=1 TO VnNumLoadedForms
  PRINT VnLoadedForm$(I) .LoadedFormName$;" ";
  PRINT VnLoadedForm$(I) .LoadedFormCaption$;" ";
  PRINT VnLoadedForm$(I) .LoadedFormVisible$
NEXT I
```

In addition to these collection specific procedures, the `'VnGetCollectionList` procedure will get the value of a specific property for all elements of any collection. For example:

```
DIM I
;
'VnGetCollectionList("Forms","Name")
;
PRINT "Names of loaded forms:"
FOR I=1 TO VnNumMembers
  PRINT VnMember$(I) .CollectionProp$
NEXT I
```

For more information on the collection procedures and the names of corresponding global variables, see each procedure's reference section.

3.4.4 Special Commands

There are several special commands that can be called by means of the `'VnCmd` procedure:

Command	Description
`Clear Form`	Deletes controls created at run time
`Load`	Loads a form into memory
`Load Picture`	Loads a graphics file
`Set Props`	Sets multiple properties at once
`Unload`	Unloads a loaded form

Table 3-3 Special Commands

The `'VnCmd` procedure is a general purpose command execution engine. The first parameter is an object, the second is a command to be performed on the object, and the third is an optional list of parameters to be passed to the command. The individual parameters are separated by the global delimiter character `_VnDelim$`, the default for which is the “pipe” symbol (|). For example:

```
'VnCmd("MyForm","Load"," ")
'VnCmd("MyForm","Unload"," ")
'VnCmd("MyForm","Clear Form"," ")
'VnCmd("MyForm","Set Props","Top=100|Left=100")
'VnCmd("MyForm.Icon","Load Picture","SWOOSH.ICO")
```

The `Load` command loads a form into memory without showing it and the `Unload` command closes a form and removes it from memory. These commands are equivalent to the VB **Load** and **Unload** statements. The `Clear Form` command is used when generating screens “on the fly.” It unloads all the control array members, except the base control (index number 0), from a particular form.

The `Set Props` command sets the values of one or more of an object’s properties. The parameter list consists of property “name=value” pairs. Each property is set to the corresponding value.

The `Load Picture` command loads a graphics file into a Picture, Icon, or DragIcon property. The third parameter is the name of the graphics file, which is loaded using the VB **LoadPicture** function.

3.4.5 Developer-Defined Commands

If `'VnCmd` doesn’t recognize the command passed in the second parameter, it passes control on to the **VnDevDef** function in the VB ***VnplDev*** module. This function is your gateway to implementing your own special-purpose commands.

To add a command, add a `Case` to the `Select` statement in the **VnDevDef** function.

```
Rem VB
' my "ErrBox" command
Case "ErrBox"
    ObjValue =  MsgBox(ObjValue, vbCritical, "Error")
```

This command could be called from an NPL program as follows:

```
'VnCmd(" ","ErrBox","Hello Vinny!")
```

3.5 Calling NPL Procedures from VB

It is possible to call almost any `PUBLIC` NPL procedure from VB. The only limitation is that you can’t use arrays as parameters to the NPL procedure. The VB **VnCallProc** function takes one or more parameters, the first of which is the name of the NPL procedure to be called. Any other parameters are passed on to the NPL procedure as its parameters. You must pass the correct number of parameters or the call will fail.

The following example shows how you would call a very poor random number generator from VB, it also shows a very poor use of this number:

```
Rem VB
VnCallProc "Random", Num
VnCallProc "Purge", Num, "START"


PROCEDURE  'Random(/POINTER Num)/PUBLIC
;NPL
;
Num=MOD(Num*1234567,20)
;
END PROCEDURE 'Random
;
PROCEDURE  'Purge(DeviceNum,FileName$8)/PUBLIC
;
SCRATCH T#DeviceNum ,FileName$
DELETE T#DeviceNum ,FileName$
;
END PROCEDURE 'Purge
```

3.6 Responding to Events

The most common place from which to call an NPL procedure is an event procedure. In fact, this is how you handle events in Visual NPL. For each event that you want to handle you do two things:

- Create a `PUBLIC` procedure in NPL
- Call the NPL procedure from the VB event procedure

For example, to handle a *Delete* button you might call an NPL procedure named `DoDelete` from VB in the following manner:

```
Private Sub Delete_ Click()
  Rem VB
  VnCallProc "DoDelete"
End Sub


PROCEDURE 'DoDelete/PUBLIC
  ;NPL
  ;
  DIM Answer
  ;
  Answer= 'VnMsgBox("Delete The Thing",
                   "Are you sure?  ",
                   _VnMbYesNo+_VnMbIconQuestion)
  IF Answer=_VnIdYes
    ; delete the thing here
  END IF
  ;
END PROCEDURE 'DoDelete
```

3.7 Error Conditions

There are several special concerns regarding calling NPL procedures from VB as follows:

1. You can only call procedures.
2. The NPL procedure must exist and be declared as `PUBLIC`.
3. The number of parameters passed must match the number of parameters expected by the NPL procedure.
4. It must be possible to convert the parameters passed into the data types expected by the procedure.
5. NPL must be executing a call to the `'VnSleep` procedure or to the `'VnCmd` procedure with the `Show, Show Modeless, Set Focus` , or `Resume` command.

For items 2 and 5 above, **VnCallProc** returns an error code. This makes it possible to handle the rare situations in which it's acceptable that an NPL procedure doesn't exist or isn't callable. All other conditions are viewed as developer errors and a message box will be displayed with the appropriate error message.

As with any program, an NPL procedure called from VB may cause an error and then stop running. This leaves you in immediate mode in NPL with no way to finish executing the NPL procedure, which means that the call to **VnCallProc** cannot complete and return to VB. However, because you are in immediate mode, this is easy to fix: type **`RETURN`**. This causes the procedure to return immediately to VB.

3.8 Other Routines

This section describes miscellaneous Visual NPL routines.

3.8.1 Controlling the NPL Window

There are four aspects of the NPL run-time window (that is, the main window created when you execute the NPL run time) that you can control.

Table 3-4 summarizes these and the NPL procedures that allow you to set and get their values.

Attribute	Set	Get
Whether or not it's visible	`'VnSetNplWndShow`	`'VnGetNplWndShow`
Position on the screen	`'VnSetNplWndPos`	`'VnGetNplWndPos`
Size of the window	`'VnSetNplWndSize`	`'VnGetNplWndSize`
Title of the window	`'VnSetNplWndTitle`	`'VnGetNplWndTitle`

Table 3-4 NPL Window Operations

In many cases, you will hide the NPL run-time window at the start of your program and never show it again:

```
'VnSetNplWndShow (_VnHide)
```

This will make the window invisible, although it is still accessible. In other words, even though the window can't be seen you can still change its position, size, and title. When you want the window to become visible again, call `'VnSetNplWndShow` again:

```
'VnSetNplWndShow(_VnShow)
```

Any changes that you made to the window while it was hidden would now be visible. When making changes to the window's position and size, remember that the coordinates and sizes are in *twips*. There are 1440 twips per inch. Also remember that the Screen object can be used to get the screen coordinates and size, which are also in twips.

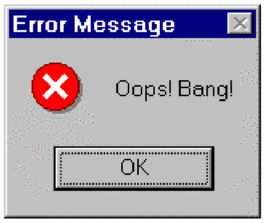
For example, to change the NPL window’s title, make the window half the size of the screen and then put it two inches from the top and left sides of the screen you would do the following:

```
DIM Width ,Height
;
Width='VnGetNum("Screen.Width")
Height='VnGetNum("Screen.Height")
;
'VnSetNplWndTitle$ ("Hello Vinny")
'VnSetNplWndSize(Width/2,Height/2)
'VnSetNplWndPos(2880,2880)
```

3.8.2 Message Boxes

Windows provides a built-in facility called the message boxThis a small window that can be used to show important messages, warnings and errors, and to ask simple questions. Windows provides a standard set of icons that can be displayed to the left of your message, as well as a standard set of buttons that can appear at the bottom of the window.

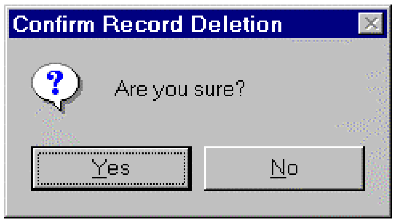
The `'VnMsgBox` function is passed the title for the window, the message to be displayed and a set of numeric flags. The flags specify which icons and buttons will appear in the window. The function returns a value that indicates which button was pressed to close the window. For example, to show a simple error message you might do the following:



```
Button='VnMsgBox("Error Message","Oops! Bang!  ",
                 _VnMbOk+_VnMbIconStop)
```

You would probably ignore the result in this call.

To do a typical deletion confirmation, you might do the following:



```
Button='VnMsgBox("Confirm Record Deletion",
                 "Are you sure? ",
                 _VnMbYesNo+_VnMbIconQuestion)
```

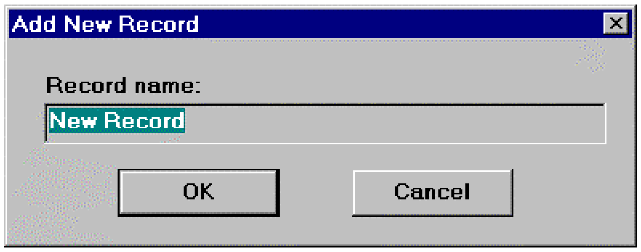
Here you would most certainly want to know what the result is, because it indicates whether or not to do the deletion:

```
IF Button=_VnIdYes
  ; delete the record
END IF
```

3.8.3 Input Boxes

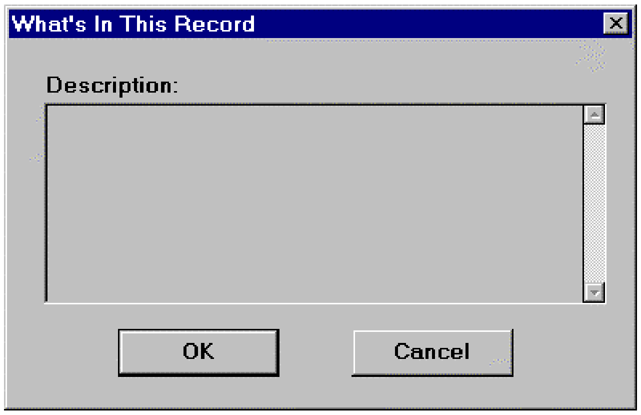
Visual NPL provides a simple window that can be used to prompt for a single input value. The `'VnInpBox$` function is passed the title for the window, the message that prompts the user, a numeric flag, and the initial value to be displayed in the input field. Set the numeric flag to 0 to use a single line input field and set it to `_VnMultiline` to use a multiple line input field. If the user clicks the *OK* button, the function returns the contents of input field; otherwise, it returns a blank string.

To prompt for a name, you might do the following:



```
Name$='VnInpBox("Add New Record","Record name:”,
                0,"New Record")
```

To prompt for a users comments, you might do the following:



```
Description$= 'VnInpBox("What's In This Record",
                  "Description:" ,_VnMultiline," ")
```

3.8.4 Closing a Form

It is *extremely important* that you handle form closing correctly. Although you may provide buttons that explicitly close (unload) a form, it is also possible to close the form using standard Windows features:

- The *Close* item on the *System* (or *Control*) menu, which is accessed with the button or icon in the upper left hand corner of the form
- The Windows 95 *Close* button (appears similar to an "X" in the upper right hand corner of the form).

If these events are not handled, it is possible for the user to close the form without your program detecting it. One way to handle this problem is by calling the provided VB **VnWakeup** procedure from the **QueryUnload** event for the form:

```
Private Sub Form_ QueryUnload(Cancel%, UnloadMode%)
    Rem VB
    VnWakeup UnloadMode
End Sub
```

The **VnWakeup** procedure checks the `UnloadMode` parameter, which indicates how the form is being closed. If it's because of a call to `'VnCmd`, then nothing is done. Otherwise, it calls the NPL `'VnWakeup` procedure. which would cause the `'VnSleep` in the form's mainline procedure to return.

3.8.5 Error Handling

Most of the NPL routines set a global variable named `VnError` to indicate whether or not they succeeded. A zero means the operation was successful, a negative value means that an error occurred. Whenever an error occurs, the `'VnErrFunc` procedure in the `"VnplDev"` module is called. This procedure sets the global variable `VnErrMsg$` to an error message that describes the problem. You can modify messages to suit your needs.

When the error message is set, the `'VnErrFunc` procedure checks another global variable named `VnErrMethod`. If this variable is set to `_VnErrCallFunc`, then a message box containing the error message is displayed. You can change this code to call your own error-handling procedure. When `'VnErrFunc` returns, the Visual NPL routine that generated the error checks the `VnErrMethod` variable itself. If the variable is set to `_VnErrSignalError`, then the routine generates an NPL run-time error code with the following statement:

```
RETURN ERROR ('VnErrNum)
```

The `'VnErrNum` function translates the value of `VnError` into an NPL error code and returns it. By default, these values start at 601 and are incremented. You can change `'VnErrNum` (in `"VnplDev"`) if this conflicts with error codes that you are already using.

A third option exists for error handling. If `VnErrMethod` is set to `VnErrSuppress`, then nothing is done with the error code; no message box is displayed and no NPL error code is returned. In this case, your program would have to manually check `VnError` after every call to an NPL routine.

To summarize, the global variable `VnErrMethod` (in `"VnplDev"`) can be set to one of the following, according to how you want Visual NPL to handle errors:

`_VnErrCallFunc`	Perform a developer-defined action
`_VnErrSignalError`	Return an NPL error code (default)
`_VnErrSuppress`	Do nothing but set `VnErrMsg$`

3.8.6 Manipulating Colors

Almost all Visual Basic objects have at least one color property. Values of color properties are of two types, expressed in different formats:

RGB Color	The number contains three 8-bit numbers indicating the intensity of the three primary colors: red, green, and blue
System Color	The number contains an index into the system table of colors

Visual NPL provides functions for creating both types of color values. It also provides a function to determine the type of a color value and separate it into its component parts. The `'VnSetRGB` function takes the three primary color intensities and returns the combined RGB color value. The `'VnSetSysColor` function takes an index and returns the system color value:

```
PRINT 'VnSetRGB(0,255,0)
PRINT 'VnSetSysColor(1)


65280
-2147483648
```

Although there are constants for the 16, standard RGB colors, you may find it useful to call `'VnSetRGB` to create other colors. On the other hand, there are only 19 system colors; constants have been created for each of these. You should use these constants directly; you should not need to call `'VnSetSysColor` . All of the color constants are documented in Chapter 7, NPL Reference.

Visual NPL also provides procedure 'VnGetColor to determine the type of a color value and separate it into its component parts. The color is passed as the only parameter and sets these global variables as a result:

VnBlueVal	Blue value if it's an RGB color
VnColor$	Name of the color if it's one of the built-in color values
VnColorSrc	Type of color (_VnRGBColor or _VnSysColor)
VnGreenVal	Green value if it's an RGB color
VnRedVal	Red value if it's an RGB color
VnRGBColorIndex	Index into VnRGBColor() and VnRGBColor$() if it's one of the built-in RGB color values
VnSysColorIndex	Index into VnSysColor() and VnSysColor$() if it's one of the built-in system color values

3.8.7 Font Translation

It is sometimes necessary to translate the NPL font into another Windows font in order for it to appear correctly with Visual Basic. This is especially true for programs in languages other than English. Visual NPL maintains an internal *font translation table*, which is used to translate string values that are passed between NPL and VB. By default, the translation table is empty, so no translation is done.

You can set the font translation table by calling the 'VnSetTran procedure with a new translation table. This table consists of pairs of characters to be translated. Within each pair, the NPL character is first, followed by the VB character.

```
'VnSetTran("AaBbCc")
```

If you pass a blank string, the font translation table is cleared and no translation is done. You can get the current translation table by calling the 'VnGetTran$ function:

```
HEXPRINT 'VnGetTran$
```

3.8.8 Detecting the NPL External Library

In some cases you may wish to detect whether or not the NPL library (**VNPL16.DLL**) is loaded before trying to use it (loading one of the modules that uses the library causes a resolve-time error if the library isn't there.) To avoid this problem, the library contains a DEFFN named 'VnDetect. When you call 'VnDetect and handle the error condition, 'VnDetect determines if the library is present or not. If it doesn't exist, then the module that uses the

For example:

```
GOSUB 'VnDetect
ERROR GOSUB 'NoLibrary
```

Note 'VnDetect can also be called by its DEFFN number, which is 32116.

3.8.9 Miscellaneous Routines

This section discusses miscellaneous routines such as translating numeric values into strings, centering a form on the screen, and getting the version number of Visual NPL.

It is often necessary in Visual NPL to translate numeric values into strings. You can call the 'VnConvNum$ function with a numeric value to get a string with no leading or trailing blanks:

```
DIM Value$10
Value$="#" & 'VnConvNum$(123) & "#"
PRINT Value$


#123#
```

Another thing you may want to do is center a form on the screen. This be done by calling the 'VnCenter procedure with the form name or object as its parameter, usually in the form's **Load** event:

```
Private Sub Form_ Load()
    Rem VB
    VnCallProc "LoadMainForm"
End Sub


PROCEDURE 'LoadMainForm
  ;NPL
  'VnCenter("MainForm")
```

```
END PROCEDURE 'LoadMainForm
```

The **VnCenter** procedure also exists on the VB side. So if the only thing you want to do in your **Load** event (or any event) is center the form, you can call **VnCenter** directly from the VB event procedure:

```
Private Sub Form_ Load()
    Rem VB
    VnCenter MainForm
End Sub
```

In addition, you can get the version number of Visual NPL by calling the `'VnGetVer$` function:

```
PRINT "Visual NPL Version ";'VnGetVer$
```

3.9 Recovery If NPL or VB Stop

As with any development project, as you are testing and modifying your program, you may find that every so often your program stops and can't be restarted from where it stopped. Ordinarily you would fix the problem and run the program again "from scratch." However, notice that, with Visual NPL, there are now two programs running: the NPL program and the VB program. As a result, when one program stops, you need to cancel the other program before you can restart everything.

3.9.1 Recovery If NPL Stops

If your NPL program stops and it is *not* executing a procedure that was called from VB, you can type the following in immediate mode:

```
'VnClose
```

This will terminate the connection with VB and the VB program itself, leaving you in a normal editing state in both NPL and VB. From there you can fix your program and then try it again. If your NPL program has multiple connections to VB applications, use `'VnCloseAll` instead of `'VnClose`.

3.9.2 Recovery If NPL Stops in a VB-Called Procedure

If your NPL program stops and it is executing a procedure that was called from VB, you must first cause the procedure call to return to VB before closing the connection between NPL and VB. You can do this by typing the following in immediate mode:

```
RETURN
```

This will cause the procedure to return to VB and the VB program to continue executing as if the NPL procedure had been successful. In some cases, continuing the program may not be acceptable. In such cases, before typing **`RETURN`**, put a break point into your VB program immediately following the call to the NPL procedure. Then, when you type **`RETURN`** in NPL, the procedure will return to VB and your VB program will stop before executing the next statement. From there you can terminate the NPL and VB programs as described in the next section.

3.9.3 Recovery If VB Stops

If your VB program stops, you can terminate the connection with NPL by typing the following in the VB Debug window:

```
VnKill
```

If the Debug window is not visible, click the *Debug Window* command under the *View* menu. The type in **`VnKill`**, making visible the NPL window, closing the connection between NPL and VB, and terminating the VB program. This leaves the system in a normal editing state in both NPL and VB.

(this page blank)

C H A P T E R 4

Change-List Programming

This chapter discusses the *change-list* method of programming with Visual NPL. It covers:

- What the change-list is and how it works
- Accessing the information in the change-list from NPL
- Adding information to the change-list from VB
- Hot controls and when they should be triggered
- Record-based forms
- Creating controls “on the fly”
- Miscellaneous programming details

4.1 What Is Change-List Programming?

Change-list programming is an attempt to bridge the gap between the NPL procedural approach and the Visual Basic event-driven approach. It is intended to be used when a large amount of NPL code can't be easily or quickly converted to the event-driven style of programming.

In effect, the change-list is a procedural wrapper put around an event-driven form. It allows the programmer to make a single call to a high-level NPL procedure, which then displays and activates a VB form. The procedure returns only when the user has pressed a button (or triggered some other control) that causes control to return to the NPL program. Any changes made to the contents of the form are returned in a global array called the change-list. This is especially useful when a form is being used to display the fields of a record.

The downside to change-list programming is that the NPL program is severely restricted with regard to the degree of control it has over the VB form that it is displaying. Furthermore, the programmer must still add VB code to respond to certain important events in order to record information in the change-list. Change-list programming views the controls on a form as the fields of a record that is being edited. Before the form is displayed, the controls are initialized to the values of the record fields. The form is then displayed and the user is allowed to edit the control values. Every time a change is made to a control, its new value is recorded in an internal array called the *change-list.*

A *hot control* is one that causes editing to stop and causes processing control to return to NPL. Typically, hot controls will be menu commands or buttons on your form. The most common cases are the *OK* and *Cancel* buttons. Eventually, the user will trigger a hot control, at which time the change-list will be returned to NPL. Depending on which control triggered the return to NPL (as recorded in the change-list,) the change-list may be processed to update the fields of the original record.

4.2 Invoking Change-List Processing

In order to use change-list programming you must use the `'VnCmd` procedure to show the form and then put NPL to sleep until a hot control is triggered. When this happens, the call to `'VnCmd` returns and the change-list is in global variables `VnChgNo` and `VnChgList$` ().

The following `'VnCmd` commands invoke change-list processing:

```
'VnCmd("FormName","Show"," ")
'VnCmd("FormName","Show Modeless"," ")
'VnCmd("FormName","Set Focus"," ")
'VnCmd(" ","Resume"," ")
```

The primary differences between the four calls are listed in this table:

Command	Shows Form?	Sets Focus To?	Clears Change-list?
`Resume`	No	Not affected	No
`Set Focus`	No	Form or control	Yes
`Show`	Yes	Default control	Yes
`Show Modeless`	Yes	Default control	Yes

Table 4.5 'VnCmd Change-List Commands

Notice that the `Show` and `Show Modeless` commands show the form and the `Set Focus` and `Resume` commands do not. This means that when processing a form you must initialize everything with a call to one of `Show` or `Show Modeless`. All subsequent processing should be done with calls to `Set Focus` and `Resume`:

```
;show the form and process user changes
'VnCmd("MainForm","Show Modeless"," ")
;
;perform user commands until the Cancel
;button is pressed or the form is closed
WHILE VnChgList$(VnChgNo) .Control$<>"Cancel"
  AND VnChgList$(VnChgNo) .Control$<>"Form Close"
  ;
  ;return to the form and process user changes
  'VnCmd("MainForm","Set Focus"," ")
WEND
```

The only difference between the `Show` and `Show Modeless` commands is in how VB creates the form. The `Show` command creates a *modal* form and the

`Show Modeless` command creates a *modeless* form. When a modal form is displayed, the user can't access the rest of your program. This is useful for windows that appear and prompt the user for input before some operation can be performed. A modeless form allows the user to access any other form currently displayed by your application. This should be used for windows that are permanently displayed.

The `Set Focus` and `Resume` commands are intended for use with a form that has already been displayed and is now processing user actions. There are two important differences between the two commands. The `Set Focus` command sets the focus to a particular form or control and clears the change-list. The `Resume` command uses the currently active form and control and doesn't affect the change-list in any way.

4.3 Accessing the Change-list from NPL

When the call to `'VnCmd` returns, the current change-list is in global variables `VnChgNo` and `VnChgList$()`. `VnChgNo` contains the number of change-list items and `VnChgList$` () contains one entry for each control that changed while the user was editing. The hot control that caused `'VnCmd` to return is the last entry in the array. The whole array is listed in order, with the most recent changes at the end.

4.3.1 Change-List Array

Each element of the change-list array is a `VnChangedList` record with the following fields:

`Flag$1`	A flag indicating what type of change-list item this is. The `Flag$` field will be one of the following characters: **C** Normal control **H** Hot control **R** Control that is mapped to an NPL record field **A** ASCII keystroke generated by a call to `VnKeyPress` **K** Keyboard code generated by a call to `VnKey` **X** Form close event generated by a call to `VnClose`
`App$8`	The name of the VB application from which the change is coming
`Form$40`	The name of the form from which the change is coming
`Control$40`	A name identifying the item that has changed
`ChgValue$40`	The changed value
`ChgValWhole$1`	A flag indicating whether or not the entire control value fits into the `ChgValue$` field. The `ChgValWhole$` field will be "Y" if the entire control value is in the `ChgValue$` field and "N" if it didn't fit. If the value is "N," then you will have to use `'VnGetAlf$` to get the full value rather than getting it from the `ChgValue$` field.

4.3.2 Hot Control

The hot control that caused `'VnCmd` to return is the last entry in the change-list. Therefore, the array will always contain at least one element and `VnChgNo` will always be at least 1. You can examine the last entry in the change-list to determine what to do next in your program:

```
'VnCmd("MainForm","Show Modeless"," ")
SWITCH VnChgList$(VnChgNo) .Control$
  CASE "OK"
    ;
  CASE "Cancel","Form Close"
    ;
END SWITCH
```

In the case of a *Cancel* button (or the form being closed), your program will probably ignore the change-list. For the *OK* button (and possibly other controls), you will want to determine if any changes were made and then process them if they exist:

```
;determine if there are any changes
```

```
IF VnChgNo>1
  ;
  ;process the changes one at a time
  FOR I=1 TO VnChgNo-1 BEGIN
    ;
  NEXT I
  ;
END IF
```

Notice that the FOR loop doesn't process the last entry in the change-list, because this is just the hot control and not actually one of the changed values.

4.3.3 Manually Accessing the Change-list

At times, it may be necessary to access the change-list in response to some event that doesn't actually return the change-list by means of 'VnCmd. The 'VnGetChgList procedure retrieves the current change-list into the global variables VnChgNo and VnChgList$(). However, there is one important difference between this change-list and the one returned by 'VnCmd. This change-list doesn't contain a hot control, which means that VnChgNo may be 0. Use the following code to determine if there are any changes and then process them if they exist:

```
;get the change-list
'VnGetChgList
;
;determine if there are any changes
IF VnChgNo>0
  ;
  ;process the changes one at a time
  FOR I=1 TO VnChgNo BEGIN
    ;
  NEXT I
  ;
END IF
```

After this is processed, you may want to clear the list so that the changes aren't applied twice. The 'VnClearChgList can be used for this purpose:

```
'VnClearChgList
```

4.4 Adding to the Change-list from VB

In order for information to be put into the change-list, you must add calls to one or more of the VB change-list routines in response to specific events. This section describes the VB routines that you must call and the circumstances for calling them.

4.4.1 Recording Changes

In order to make use of the change-list, a control must notify Visual NPL whenever its value changes. This is done by calling the `VnChg` procedure from a VB event that indicates that something has changed, usually **Change**, **Click**, or **DblClick**. Which event(s) to use depend on the type of control and how you are trying to use it.

The **VnChg** procedure simply sets an internal flag that indicates that the control's value has been changed. In order to actually add the new value to the change-list, you must call the **VnChk** procedure from a VB event that indicates that editing of the control is complete, typically, the **LostFocus** event. The **VnChk** procedure checks the internal flag, if it's not set, then nothing is done. When the flag has been set by an earlier call to **VnChg**, the value of the control is copied into the change-list array and the element's `Flag$` field is set to **“C.”**

With some controls, particularly buttons, you will want to call **VnChk** right after calling **VnChg**. This is useful when the control is initiating some sort of action that should be performed immediately. This is the case with clicking a command button or spin button and double-clicking some other controls, particularly lists.

This table shows the VB events that can be used for **VnChg** and **VnChk** for the standard VB controls.

Control	VnChg	VnChk
Check box	Click	LostFocus
Combo box	Change, Click or DblClick	LostFocus or DblClick
Command button	Click	Click
Directory list box	Change or Click	LostFocus or DblClick
Drive list box	Change	LostFocus
File list box	Click or DblClick	LostFocus or DblClick
Grid	Click	LostFocus
Horizontal scroll bar	Change	LostFocus
List box	Click or DblClick	LostFocus or DblClick
Option button	Click	LostFocus
Spin button	SpinDown and SpinUp	SpinDown and SpinUp
Text box	Change	LostFocus
Vertical scroll bar	Change	LostFocus

Table 4-6 VnChg and VnChk Events

For example, for a text box control, you would fill in the VB **Change** and **LostFocus** events in this way:

```
Private Sub Address_ Change()
    Rem VB
    VnChg
End Sub

Private Sub Address_ LostFocus()
    VnChk
End Sub
```

For a button control, you would fill in the **Click** event in this way:

```
Private Sub Cancel_ Click()
    Rem VB
    VnChg
    VnChk
End Sub
```

4.4.2 Creating Hot Controls

There are two ways to make a normal control into a hot control

- Call VB procedure **VnHot** instead of calling **VnChk.**
- Put “Hot” into the Tag property of the control.

Use one method or the other; do not use both. Using **VnHot** has the advantage of simplicity; you type in **VnHot** instead of **VnChk**. The **VnHot** procedure works exactly like the **VnChk** procedure with one exception. If the control’s value is copied to the change-list, the change-list is then returned to NPL. For example, the **Click** event for an *OK* button would look similar to this:

```
Private Sub OKButton_ Click()
    Rem VB
    VnChg
    VnHot
End Sub
```

Using the Tag property has the advantage of flexibility. You can change a control into a hot control (or vice versa) at run time just by setting the Tag property from NPL:

```
'VnSetAlf("MainForm.SomeControl.Tag","Hot")
```

When setting the Tag property, you must still call **VnChk**. When the **VnChk** procedure copies a control’s value to the change-list, it also checks the control’s Tag property. If it contains the word “Hot”, then the change-list is returned to NPL.

Regardless of which method is used, the last entry in the change-list (the hot control) has its `Flag$` field is set to **`"H"`**.

4.4.3 Menu Commands

The **VnMenuClk** procedure is used to add an entry for a menu command to the change-list and then return the change-list to NPL. This procedure acts in a similar manner to a hot control in that it forces program control to return to NPL. It takes one parameter—the name you want returned in the `Control$` field of the change-list entry. This name can be anything you want as the following example shows.

```
Private Sub FileMenu_ Click(Index%)
    Rem VB
    Select Case Index
        Case 1
            VnMenuClk "FileNew"
        Case 2
            VnMenuClk "FileOpen"
        Case 3
            VnMenuClk "FileSave"
        Case 4
            VnMenuClk "FileSaveAs"
        Case 5
            VnMenuClk "FilePrint"
        Case 6
            VnMenuClk "FileExit"
    End Select
End Sub
```

4.4.4 Closing a Form

It is *extremely important* that you handle form-closing correctly. Although you may provide buttons that explicitly close (unload) a form, it is also possible to close the form using standard Windows features:

- The *Close* command on the *System* (or *Control*) menu, which is accessed with the button or icon in the upper left corner of the form
- The Windows 95 *Close* button (this appears similar to an “X” in the upper right corner of the form

If these events are not handled, it will be possible for the user to close the form without your program detecting it. One solution is to call the VB **VnClose** procedure from the **QueryUnload** event for the form:

```
Private Sub Form_ QueryUnload(Cancel%, UnloadMode%)
    Rem VB
    VnClose UnloadMode
End Sub
```

The **VnClose** procedure checks the `UnloadMode` parameter, which tells how the form is being closed. If it's because of a call to `VnCmd` in NPL, nothing is done. If it's being closed in some other way, then a special change-list entry is added and the change-list is returned to NPL. The special entry will have the `Flag$` field set to `"X"` and the `Control$` field set to **`"Form Close"`**. This procedure acts similarly to a hot control in that it forces program control to return to NPL.

4.4.5 Keyboard Handling

It is sometimes necessary to process each keystroke that a user types. There are two different types of keyboard events in Visual Basic. The **KeyDown** and **KeyUp** events are triggered when any key is pressed or released, including function keys and special keys. These procedures are passed a numeric code that identifies the key. The **KeyPress** event is triggered whenever an ASCII key is released. The parameter passed to this procedure is the numeric code for an ASCII character.

If you want to respond to any or all of these keyboard events, call the **VnKey** procedure from the **KeyDown** and **KeyUp** events, passing it the parameters that are passed to the event:

```
Private Sub Address_ KeyDown(KeyCode%, Shift%)
    Rem VB
    VnKey KeyCode, Shift
End Sub

Private Sub Address_ KeyUp(KeyCode%, Shift%)
    Rem VB
    VnKey KeyCode, Shift
End Sub
```

Call the **VnKeyPress** procedure from the **KeyPress** event, passing it the parameter that is passed to the event:

```
Private Sub Address_ KeyPress(KeyAscii%)
    Rem VB
    VnKeyPress KeyAscii
End Sub
```

In the preceding examples, the keyboard events are all for a control named ***Address.*** It is also possible for a form to get keyboard events by setting its KeyPreview property to True. In this case the form's keyboard events will be triggered before the control's events. This allows you to do keyboard processing that is common to all controls.

Whether you call the **VnKey** and **VnKeyPress** procedures from a form or an event, each procedure passes its parameters back to NPL as a separate, single-entry change-list, without affecting the current change-list. The change-list entry will have the `Flag$` field set to **`"K"`** if **VnKey** was called and **`"A"`** if **VnKeyPress** was called.

When the change-list is processed on the NPL side, some special global variables are set before it is returned to your program:

`VnKeyin$`	Set to "True" (normally "False") to indicate that a keyboard change-list is being returned to NPL
`VnKeyinMode$`	Set to `"K"` if VnKey was called and `"A"` if VnKeyPress was called

For the **VnKey** procedure, the following global variables are set:

`VnKey$`	The KeyCode parameter from VB
`VnShift`	The Shift parameter from VB
`VnKeyType$`	HEX(00) for an ASCII key and HEX(01) for a special key

For the **VnKeyPress** procedure, the following global variables are set:

`VnKey$`	The KeyAscii parameter from VB

Because the **VnKey** and **VnKeyPress** calls don't modify the real change-list, the next call to `'VnCmd` should use the `Resume` command in order to leave it intact.

4.5 Record-Based Forms

With standard change-list programming you must make several calls to initialize the values of the controls on a form. This is done with the `'VnSetAlf` and `'VnSetNum` procedures. Then, when editing is complete, you must retrieve the new values one at a time from the change-list. Although this method is effective, for the sake of performance and simplicity, you may want to be able to set and get the values of the controls on a form with one call. Record-based forms provide this capability.

4.5.1 Setting and Getting Records

The `'VnSetRec` procedure is passed a form, the name of a `PUBLIC` NPL record, and the record itself. The procedure goes through the fields of the record and sets the controls on the form to the values stored in the record. When your program has initialized the controls, it calls `'VnCmd` in the normal manner.

When 'VnCmd returns, the change-list is still used to determine if any changes were made. However, rather than retrieving the changes from the change-list, they are retrieved by calling 'VnGetRec. This procedure is passed a form and the name of a PUBLIC NPL record. It retrieves the fields in the record from the controls on the form and then returns the new record.

```
;set the controls from the record
'VnSetRec("MainForm","MainRecord",Main$)
;
;show the form modelessly,
;allow the user to edit the controls,
;return when a Hot control is triggered
'VnCmd("MainForm","Show Modeless"," ")
;
;if the OK button was pressed AND something
;has changed then process the change-list
IF VnChgList$(VnChgNo) .Control$="OK"
   AND VnChgNo>1
   ;
   ;get the record from the controls
   Main$='VnGetRec$("MainForm","MainRecord")
END IF
```

Notice that, when the return value from 'VnGetRec is assigned to Main$, it overwrites the original record. This can cause a problem when the record contains fields that are not on the form. These fields will be set to blanks by the assignment statement. To avoid this problem, use the 'VnGetRecSubset procedure instead of 'VnGetRec. Whereas 'VnGetRec is a function that returns a new record, 'VnGetRecSubset is a procedure that is passed a record to be updated. Any fields in the record that are not on the form are left intact.

```
'VnGetRecSubset("MainForm","MainRecord",Main$)
```

instead of

```
Main$='VnGetRec$("MainForm","MainRecord")
```

4.5.2 Mapping Fields to Controls

In order for the `'VnSetRec` and `'VnGetRec` calls to work, you must map each field in the NPL record to a control on the VB form. This is done by setting the Tag properties of the controls to the corresponding record and field names.

For example, given the following NPL record:

```
RECORD Person
  FIELD Name$40
  FIELD Address$40
  FIELD Salary
END RECORD
```

you would set the Tag properties of the appropriate controls to:

```
Person.Name
Person.Address
Person.Salary
```

Note that the dollar sign ($) on the end of string field names is omitted.

4.6 Creating Controls On The Fly

Most Visual NPL programming is done by creating forms at design time and then working with these preconstructed forms at run time. However, it is also possible to create forms and controls at run time. The controls "on the fly" routines treat a VB form as a text based screen with a certain number of rows and columns. Most of the routines are passed a row and column number (among other things), where they will "print" their information (or controls).

When "printing" to a form, the form's AutoRedraw property must be set to True, otherwise the form's contents will disappear whenever the form is minimized or overlaid by another form.

4.6.1 Row and Column Mapping

Obviously, the row and column numbers must be converted into the twips coordinates used by Visual Basic. The number of rows and columns is defined by calling the `'VnSetRowsCols` procedure.

```
'VnSetRowsCols("VnplChar",20,60)
```

It is important to remember that you must call this routine each time you want to print to a different form, because the mapping information is not saved anywhere internally. So, if you have multiple forms being displayed, you must call `'VnSetRowsCols` each time the user moves to a new form (if you intend on printing to that form).

Furthermore, you must call this routine and redraw everything on a form whenever it is resized. Otherwise, the row and column sizes will be different and any new printing won't align properly with what's already on the form.

4.6.2 Printing to a Form

The `'VnPrintAt` procedure can be used to print text directly on a form. It is passed the form to be printed on, the row and column to print at and the text to be printed:

```
'VnPrintAt("MainForm",12,25,"Hello there!")
```

The `'VnPrintBox` procedure can be used to print a rectangle of a specific color on a form. Pass it the form on which to print, the row and column at which to print, and the height, width and color of the box:

```
'VnPrintBox("MainForm",2,20,5,40,_VnLightGray)
```

You can also print several things to the internal print buffer (`VnPrint$`) and then send the buffer to VB for printing. The `VnPrint$` global variable declared in `"VnplDev"` is large enough to handle the text for a 24–row by 80–column screen. If you plan on using more rows and columns you may have to enlarge this buffer.

To print to the buffer, use the NPL `PRINT TO` and `PRINTUSING TO` statements. However, the NPL AT statement won't work when sending the buffer to VB. Instead, use the `'VnAt$` function, which is passed a row and a column number and returns the characters necessary to do row/column positioning in VB. When finished printing to the buffer you can call the `'VnPrintTo` procedure to send the buffer to VB. This procedure is passed one parameter—the form to print on.

```
PRINT TO VnPrint$;'VnAt$(2,4);"Name:"
PRINT TO VnPrint$;'VnAt$(4,4);"Address:"
PRINT TO VnPrint$;'VnAt$(10,4);"Phone:"
'VnPrintTo("MainForm")
```

A third procedure, `'VnInputScreen`, can be used to copy all or part of the NPL screen to a VB form. Pass this procedure the form on which to print, the row and column at which to print, and the height and width of the rectangular area to copy to VB. This procedure does the equivalent of an `INPUT SCREEN` from the NPL screen and a `PRINT SCREEN` to a VB form.

```
PRINT AT(2,4);"Name:"
PRINT AT(4,4);"Address:"
PRINT AT(6,4);"Phone:"
'VnInputScreen("MainForm",0,0,10,40)
```

The printing routines are not limited to printing on a form. They can also be used with a picture box object and the printer. Everything except the `'VnPrintBox` routine can also be used on the VB Debug window.

4.6.3 Base Controls

Creating new controls at run time (*controls "on the fly"*) works by making duplicates of a set of base controls. You must create a form in VB with one *base control* for every type of control you need on the form. For example, if the form only uses labels, text boxes, and buttons, then you would need a base label, a base text box, and a base button. Each base control should have its Index property set to 0 and its Visible property set to False. You can name them whatever you want, but usually the type name is used (***Label***, ***TextBox***, ***CommandButton***).

Setting the Index property to 0 causes VB to create a single-element array where element 0 is the control you create. When you create a new control at run time, Visual NPL makes a copy of the base control by enlarging (using `REDIM`) the control array. The new element is an exact duplicate of the base control except for the Index property. It will be set to the new control's position in the control array.

Note The ***VnplChar*** form contains base controls for all of the standard Visual Basic controls. You can use this form in your programs.

4.6.4 Registering Control Names

For every unique base control name, you must register the control array in the **VnSetCtrl** function in the ***VnplDev*** module in VB. This means adding a new `Case` to the `Select` statement in **VnSetCtrl**. The text of the ***`Case`*** should be the name of the control array and the single line of code for the `Case` should set the `Ctrl` parameter to element Index of the control array. For example, for a control array named ***NewControl***, the VB `Case` statement would look similar to this:

```
Case "NewControl"
    Set Ctrl =  Obj.NewControl(Index)
```

The default `Select` statement contains `Case` statements for each of the standard Visual Basic controls (that is, the ones on the ***VnplChar*** form).

4.6.5 Creating Controls

The NPL `'VnCreateCtrls` procedure creates new controls at run time. It is passed the form on which to create controls, the name of the base control, and the number of new controls to create. For example, to create three input fields, their captions, and an *OK* and a *Cancel* button, you would do the following:

```
'VnCreateCtrls("VnplChar","TextBox",3)
'VnCreateCtrls("VnplChar","Label",3)
'VnCreateCtrls("VnplChar","CommandButton",2)
```

This just creates the controls; it doesn't show them. To do that, you must "print" them to the form. Pass the `'VnPrintCtrl` procedure:

- The control to be printed
- The row and column to print at
- The height and width of the control
- The value to be assigned to control's default property
- A list of properties and their values

The `'VnPrintCtrl` procedure initializes a control and then shows it:

```
'VnPrintCtrl("VnplChar.Label(1)",2,2,1,1,"Name:"," ")
'VnPrintCtrl("VnplChar.Label(2)",4,2,1,1,"Address:",
             " ")
'VnPrintCtrl("VnplChar.Label(3)",6,2,1,1,"Phone:",
             " ")
;
'VnPrintCtrl("VnplChar.TextBox(1)",2,11,1,40,
             " ","MaxLength=40,TabIndex=1")
'VnPrintCtrl("VnplChar.TextBox(2)",4,11,1,40,
                " ","MaxLength=40,TabIndex=2")
'VnPrintCtrl("VnplChar.TextBox(3)",6,11,1,10,
                " ","MaxLength=10,TabIndex=3")
;
'VnPrintCtrl("VnplChar.CommandButton(1)",9,13,2,10,
                " ","Caption=OK ,TabIndex=4,Tag=Hot")
'VnPrintCtrl("VnplChar.CommandButton(2)",9,28,2,10,
                " ","Caption=Cancel ,TabIndex=5,Tag=Hot")
```

4.6.6 Destroying Controls

When you unload a form, all the controls on the form are destroyed. However, with controls "on the fly," you may want to create one *base form* (such as ***VnplChar***) and use it to show many other forms. In this case, you would need to clear the form and recreate the controls on it. The `Clear Form` command of the `'VnCmd` procedure does that, destroying all the controls on the form except for the base controls.

```
'VnCmd("VnplChar","Clear Form"," ")
```

To destroy individual controls, pass the control to the `'VnDestroyCtrl` procedure. For example:

```
'VnDestroyCtrl("VnplChar.Label(1)")
```

C H A P T E R 5

Demo Programs

This chapter discusses the demonstration programs that come with Visual NPL. These programsare meant to be used as a guide to various Visual NPL features. They include:

- A simple change-list program
- A simple event-driven program
- A large demo program that contains several smaller demo programs, each of which demonstrates a specific Visual NPL feature

5.1 Hello (Change-List)

The ***Change-list Hello*** program is a simple example that demonstrates the basics of change-list programming.

It calls the **VnChg** and **VnChk** procedures from the **Click** event of the *OK* button. Because the Tag property for this button is set to "Hot", the **VnChk** procedure causes the change-list to be passed back to the NPL program, which in turn causes the program to end. The program also calls the **VnClose** procedure from the form's **QueryUnload** event. This handles the case where the user closes the form from the system menu or by clicking the *Close* button in the upper right corner of the window (represented by an "X") in Windows 95.

5.2 Hello (Event-Driven)

The ***Event Driven Hello*** program is a simple example that demonstrates the basics of event-driven programming.

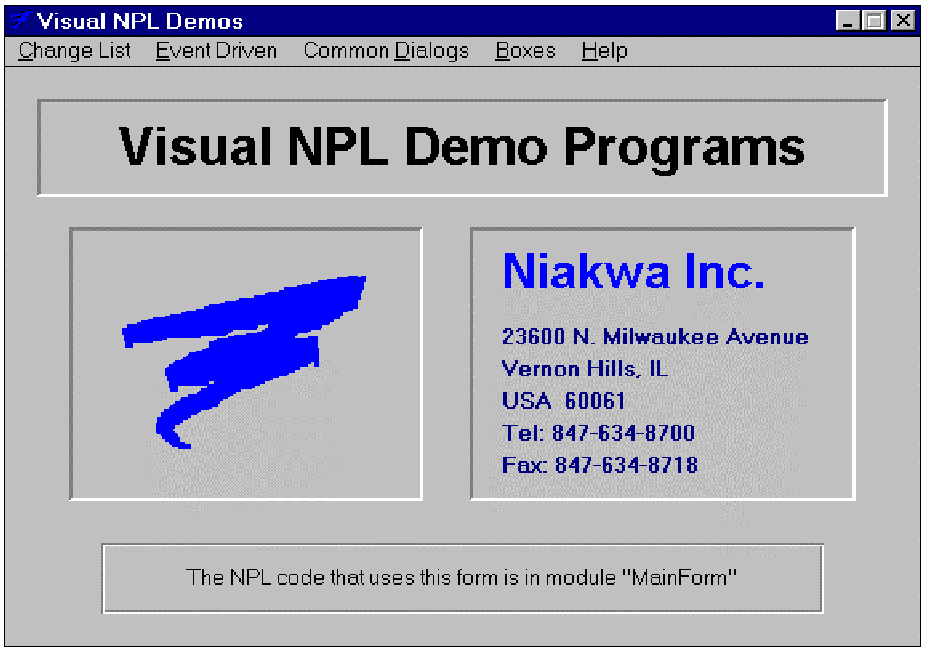
It uses the **VnCallProc** procedure in the **Click** event of the *OK* button to call the NPL `'VnWakeup` procedure. This causes the mainline of the NPL program to return from the `'VnSleep` call and then end the program. The program also calls the **VnWakeup** procedure from the form's **QueryUnload** event. This handles the case where the user closes the form from the system menu by clicking the *Close* button in the upper right corner of the window (represented by an "X") in Windows 95.

5.3 Demos

The ***Demos*** program consists of several subprograms, each of which demonstrates one or more Visual NPL concepts. The subprograms are organized into four groups, each of which can be accessed from a top-level menu on the main window:

Change-list	Change-list programs
Event Driven	Event-driven programs
Common Dialogs	Common dialog examples
Boxes	Message and input boxes

The menu commands under each top-level menu correspond to the individual subprograms for each group. The code that controls the main window and launches all of the other subprograms is also a subprogram itself. It's a good example of how to use menus and can be found in the NPL "MainForm" module in **DEMOS.NPL** and the VB ***MainForm*** form in files **MAINFORM.FRM** and **MAINFORM.FRX**. The "MainForm" module also contains the code for the Common Dialog examples and Boxes examples.



The NPL code that uses this form is in module "MainForm"

Figure 5.2 Demos Program Main Window

The change-list and event-driven subprograms are all stand-alone Visual NPL programs when combined with the NPL "START" and "END" modules. Each of these subprograms consists of a main NPL module and a VB form. In some cases there may also be additional NPL code and/or data modules involved. This modularization should make it easier to insert pieces of the ***Demos*** program into your own programs.

5.3.1 Change-List Programs

The change-list demos show how to use the change-list programming features including:

- Processing menu commands
- Creating forms and controls at run time
- Printing to a VB form

- Processing input/output (I/O) one field at a time
- Processing I/O with all fields at once
- Terminating the program

The demos are accessed with the following menu commands:

Old Fashioned NPL Record I/O	Uses the NPL window to do record I/O.
Generate VB Screens at Run Time	Generates a VB window from code at run-time.
Transfer NPL Screen Text to VB	Prints to the NPL window and then copies it to a VB form.
Print Text on VB Form	Prints text directly to a VB form.
Field I/O Using a VB Form	Does record I/O, accessing the controls on a VB form one at a time.
Record I/O Using a VB Form	Does record I/O, accessing the controls on a VB form as a group that makes up a record.
Combo Boxes and Record I/O	Does record I/O, accessing the controls on a VB form as a group that makes up a record, while using combo boxes for some of the fixed-choice fields.
Exit	Terminates the program.

5.3.2 Event-Driven Programs

The event-drivendemos show how to use event-driven programming features including:

- Processing menu commands
- Showing a progress meter
- Dealing with color
- Viewing images
- Dealing with objects (forms, controls, and properties)
- validating input data,
- Setting up and using the printer
- Terminating the program

The demos are accessed with the following menu commands:

Colors	Selecting and setting colors.
Progress Bar	Showing and cancelling a progress indicator.
Image Viewer	Selecting drives, directories, and files, and showing images.
Database Access	Using the data control and data bound controls.
Unbound Data Grid	Using a grid to show data records (see the note that follows).
Objects - Set Properties	Accessing objects and setting their properties.
Objects - Inspect	Accessing objects and viewing their properties in a grid.
Validation - Post	Validating input when the OK button is clicked.
Validation - Keystroke	Validating input whenever a key is pressed.
Validation - Control	Validating input using a self-validating control.
Validation - Real-Time	Validating input on a field by field basis.
Printer - Setup	Setting up the printer.
Printer - Test	Printing text.
Exit	Terminate the program.

Note The standard grid that comes with VB is a limited version that may exhibit instability. As a result, the ***Unbound Data Grid*** demo may behave unpredictably. If you need to use a grid it is *highly recommended* that you buy a full-featured grid control from a third party. However, you may still view the grid demo for an illustration of how to use a grid.

5.3.3 Common Dialogs

The common dialog demos show how to use the VB support for Windows' built-in common dialogs (sometimes called *pop-up windows*), which provide access to several common Windows operations:

- Selecting file names
- Selecting colors
- Selecting printers
- Selecting fonts
- Invoking the Help system

The CommonDialog control is used by setting various properties (depending on the action to be performed) and then calling one of its **Show** methods:

ShowOpen	Select a file to be opened
ShowSave	Select a file to which to save
ShowColor	Select a color
ShowPrinter	Select and configure a printer
ShowFont	Select and configure a font
ShowHelp	Invoke the Help system

The CommonDialog control is a control that has no visual interface and that exists on the ***MainForm.*** It will appear at design time but not at run time. The code that uses the control is in procedure `'CommonDialog` in the NPL `"MainForm"` module.

5.3.4 Boxes

The box demos show how to use the `'VnMsgBox` and `'VnInpBox$` functions. The `'VnMsgBox` function can be used to show simple error or information messages as well as to prompt for the following types of user responses:

- *OK* or *Cance*l
- *Abort, Retry*, or *Ignore*
- *Yes* or *No*
- *Yes, No,* or *Cance*l
- *Retry* or *Cancel*

The `'VnInpBox$` function is used to prompt for a single-line or a multiline text value. The user is prompted with a message, an input field, and the *OK* and *Cancel* buttons. If *OK* is pressed, then the input field is returned; if *Cancel* is pressed, then a blank string is returned. The code for the box demos is in procedure `'Boxes` in the `"MainForm"` module.

(this page blank)

C H A P T E R 6

Distributing Visual NPL Programs

This chapter contains a discussion of what needs to be done when installing a Visual NPL application on a user's system. It covers the following:

- An introduction to what needs to be done to set up the user's system
- How to use the Visual Basic Setup Wizard to create a setup program
- A description of the problems associated with deciding not to use a setup program
- How to register OCX controls
- A list of required support files

6.1 Installation Considerations

There are many things to consider when installing a Windows program on a user's computer, including the following:

- Determining the destination directory and creating it, if necessary
- Insuring that there is enough disk space on the destination drive
- Copying and possibly decompressing files, making sure not to overwrite a newer file that already exists on the user's computer
- Managing diskette switches and prompting the user as appropriate
- Registering DLLs and OCXs
- Creating folders, icons, and *Start* menu commands

Usually all of this is managed by an installation (or setup) program that the user runs from the first diskette. You can create this type of program with the Visual Basic Setup Wizard.

6.2 Using the Setup Wizard

The Setup Wizard is a Windows program that performs seven simple steps in order to produce an installation/setup program and the compressed files to be installed. Here's what happens in each of the seven steps:

1. The wizard prompts you to enter the name of the Visual Basic project file (that is, **DEMOS.VBP**).
2. The project file is analyzed to determine the files that need to be installed.
3. The wizard prompts you to choose the installation medium—diskettes or hard-disk drive.
4. You can add any OLE servers that your program requires but that weren't listed in the project file.
5. You can remove any unnecessary OCXs or DLLs.
6. You must select the type of installation to perform: normal application (**.EXE**) or OLE server.
7. You can add any other files to the list of files to be installed.

In general, you usually don't need to specify any entries for steps 2, 4, 5, and 6. You can leave them set to their defaults. This is because step 2 is completely automated; it determines which OLE servers (step 4) and OCX controls (step 5) are used by your program as well as what type of program it is (step 6). In other words, steps 4, 5, and 6 are only provided as safeguards in case something is missed in step 2.

Furthermore, steps 1 and 3 are easy choices that should only take a few moments. This leaves only step 7 in which you have to do any real work. This is where you do the following:

- Add the names of your NPL disk images, including **`VNPL.NPL`**
- Add the names of your data files
- Add **`VNPL16.DLL`**
- Add **`VNPL16.DLL`**'s dependent files (listed as follows)

After completing this step, clicking the *Finish* button will create the installation files and copy them to the chosen installation medium.

When this is done, you can save the setup configuration as a template. Then, the next time you need to produce the installation files, all you need to do is open the template in step 1 (instead of entering the name of the project file) and click the *Finish* button.

6.3 Distributing Visual NPL Without a Setup Program

Although it is possible to distribute a Visual NPL application without using a setup program, it is not recommended. Along with the basic operation of copying files, you must consider several factors when installing a Windows program:

- All OCXs and DLLs should be installed into the **`Windows\System`** directory.
- An older DLL (based on the file date or an internal version number) should not be installed over a newer one.
- All OCXs must be "registered" before they can be used by any program, including yours.

Although copying the files into the specified directory is easy, it can be difficult to determine the internal version numbers of files being installed in order to avoid copying older versions over newer ones. Using a setup program can reduce the complexity of the installation process and eliminate the potential for problems by detecting and installing the appropriate file versions.

In addition to the version number problem, you must register all of your controls so that they are usable. As described in the next section, this involves installing a registration program and calling it for each OCX.

Even if you don't use a setup program of any kind, the Setup Wizard that comes with Visual Basic is an excellent tool for determining which files need to be installed.

6.4 Registering OCXs

One of the most important things that an installation/setup program must do is to register the OCXs that it installs. Registration is the act of telling Windows about an OCX (or program) by adding some special entries to the Windows registry, and an OCX can't be used until it is registered.

Almost all OCX controls contain code to register themselves. The setup program loads each OCX and invokes its registration code. If you aren't using a setup program, then you can use the **`REGSVR.EXE`** program in the **`Windows`** directory. For each OCX that you need to register, call the program as follows:

```
REGSVR /s DllName
```

for example:

```
REGSVR /s VNCON16.OCX
```

The **/s** parameter tells **REGSVR** to run in silent mode; otherwise, it shows a message box when it is done.

6.5 Required Support Files

The following support files are required by Visual NPL:

VNCON16.OCX	**VNPL16.DLL**
VNPL.NPL	

The following support files are required by **VNPL16.DLL**:

MFCO250.DLL	**MFC250.DLL**

The following support files are required by Visual Basic:

CTL3DV2.DLL	**VAEN21.OLB**
OC25.DLL	**VB40016.DLL**
OLE2.DLL	

The following support files are required by **OLE2.DLL**:

COMPOBJ.DLL	**OLE2.REG**
OLE2CONV.DLL	**SCP.DLL**
OLE2DISP.DLL	**STDOLE.TLB**
OLE2PROX.DLL	**STORAGE.DLL**
OLE2NLS.DLL	**TYPELIB.DLL**

(this page blank)

C H A P T E R 7

NPL Reference

This chapter contains:

- Detailed descriptions of each of the NPL constants
- Definitions for each of the NPL records used to retrieve various lists
- Descriptions of each of the NPL variables used to return information from some of the NPL subroutines
- A list of the NPL subroutines categorized by the type of operation they perform
- Definitions and detailed descriptions of each of the NPL subroutines in alphabetical order

7.1 Constants

This section discusses the most important constants defined in the `"Vnpl"` and `"VnplDev"` modules. It also gives an overview of the other constants that are available.

7.1.1 VnSys (VNPL.NPL Device Number)

The `_VnSys` constant is defined in `"VnplDev"` and is used to determine which device refers to the **`VNPL.NPL`** disk image.

It should be used in modules that use Visual NPL as follows:

```
INCLUDE T "VnplDev"
INCLUDE T#_VnSys,"Vnpl"
```

The default value is 1.

7.1.2 VnDelim$ (Parameter Delimiter)

The `_VnDelim$1` constant is defined in `"VnplDev"` and is used as the delimiter whenever a list of items is passed between NPL and VB.

This value's main use is to separate parameters passed to `VnMethod`. Ideally, the delimiter character should adhere to the following criteria:

- Is not found in normal data items
- Is directly enterable from a computer keyboard
- Has the appearance of a separator, rather than of data

The default value is a vertical bar or "pipe" character (|).

7.1.3 VnStrRefSize (Minimum /POINTER String Parameter Size)

The `_VnStrRefSize` constant specifies the minimum size of a string parameter passed from ***VnCallProc*** in VB into a `/POINTER` string parameter in NPL.

If the string being passed is smaller than the specified minimum, the DLL will allocate a buffer of this size, fill it with the passed value, and then pass it to the NPL procedure.

The default value is 100.

7.1.4 Maximum Number of Controls and Properties

The `_VnMaxCtrlsPerForm` and `_VnMaxPropFields` constants specify the maximum number of controls returned by `'VnGetFormCtrlList` and the maximum number of control properties returned by `'VnGetPropInfoList`, respectively.

These values are used when doing a `MAT SORT` on the results produced by the corresponding routines.

The default value for `_VnMaxPropFields` is 80 and the default value for `_VnMaxCtrlsPerForm` is 150.

7.1.5 Key Translation Strings

The following constants are used to translate Windows keystrokes from the **KeyDown** and **KeyUp** events into NPL keyboard codes:

`_VnNplShiftKbrd$21`	`_VnUnshiftAlphaKbrdTran$52`
`_VnNplUnshiftKbrd$21`	`_VnWndUnshiftKbrd$21`
`_VnShiftAlphaKbrd$26`	

7.1.6 Error Handling Flags and Error Codes

The following are error-handling methods:

`_VnErrCallFunc`	`_VnErrSignalError`
`_VnErrSuppress`	

The following are error codes:

`_VnErrBadAppNum`	Invalid application number or application not open
`_VnErrBadCommand`	Invalid command name
`_VnErrBadControlName`	Invalid control name (or control name not registered in `VNPLDEV.BAS`)
`_VnErrBadFormName`	Invalid form name (or form name not registered in `VNPLDEV.BAS`)
`_VnErrBadIflags`	Invalid input box flags
`_VnErrBadMflags`	Invalid message box flags
`_VnErrBadMode`	Invalid show window mode
`_VnErrBadObjectName`	Invalid object name
`_VnErrBadParam`	Invalid parameter
`_VnErrBadParamCount`	Wrong number of parameters
`_VnErrBadPrintRowOrCol`	Invalid row or column number

`_VnErrBadPropertyName`	Invalid property name
`_VnErrBadRecFieldName`	Field name from VB tag matches no `PUBLIC FIELD` name
`_VnErrBadValueLen`	Invalid value length
`_VnErrBadVersion`	Version-number mismatch between `VNPL16.OCX` and `VNPL16.DLL`
`_VnErrCouldNotAccess`	Couldn't access object, property, or method
`_VnErrExeFailed`	The `.EXE` file couldn't be run or VB isn't running
`_VnErrMissingSeparator`	Missing parameter separator (delimiter)
`_VnErrNoConnect`	Communications couldn't be established with the VB application
`_VnErrNoMemory`	Memory for the communications buffer couldn't be allocated
`_VnErrPastDropDeadDate`	This product's limited license has expired
`_VnErrPosFailed`	Invalid NPL Window position
`_VnErrSizeFailed`	Invalid NPL Window size
`_VnErrStrTooLong`	A string parameter is too long
`_VnErrTooManyApps`	The maximum number of open applications are already open
`_VnErrValueNotNumeric`	The retrieved string couldn't be converted to a number
`_VnErrVbError`	Visual Basic error message

The following are system variables for capacity:

`_VnMaxApps`	`_VnObjLen`
`_VnMaxValueLen`	`_VnObjNameLen`

7.1.7 Message Box and Input Box Flags

The following flags describe the buttons that will appear in a message box and a mask for extracting this flag from the combined flags:

`_VnMbAbortRetryIgnore`	`_VnMbYesNoCancel`
`_VnMbOk`	`_VnMbRetryCancel`
`_VnMbOkCancel`	`_VnMbTypeMask`
`_VnMbYesNo`	

The following flags describe the icon that will appear in a message box and a mask for extracting this flag from the combined flags:

`_VnMbIconExclamation`	`_VnMbIconQuestion`
`_VnMbIconInformation`	`_VnMbIconStop`
`_VnMbIconMask`	

The following flags describe the button that will be the default button in a message box and a mask for extracting this flag from the combined flags:

`_VnMbDefButton1`	`_VnMbDefButton3`
`_VnMbDefButton2`	`_VnMbDefMask`

The following flags describe the modality of a message box:

`_VnMbApplModal`	`_VnMbTaskModal`
`_VnMbSystemModal`	

The following flag causes a message box not to have the focus when it first appears:

`_VnMbNoFocus`

The following return values indicate which button was pressed in a message box:

`_VnIdIgnore`	`_VnMbIdCancel`
`_VnIdNo`	`_VnMbIdOk`
`_VnIdYes`	`_VnMbIdRetry`
`_VnMbIdAbort`	

This flag indicates whether or not an input box is multiline:

`_VnMultiline`

7.1.8 Window Show Modes

These constants are used by `'VnSetNplWndShow` and `'VnSetNplWndShow` :

`_VnHide`	`_VnShow`

7.1.9 Color Constants

These constants tell the type of color returned from `'VnGetColor` :

`_VnRGBColor`	`_VnSysColor`

The RGB color constants are:

`_VnBlack`	`_VnLightGreen`
`_VnBlue`	`_VnLightMagenta`
`_VnCyan`	`_VnLightRed`
`_VnGray`	`_VnLightYellow`
`_VnGreen`	`_VnMagenta`
`_VnLightBlue`	`_VnRed`
`_VnLightCyan`	`_VnYellow`
`_VnLightGray`	`_VnWhite`

The system color constants are:

_VnActiveBorder	_VnInactiveBorder
_VnActiveTitleBar	_VnInactiveTitleBar
_VnAppWorkspace	_VnMenuBar
_VnButtonFace	_VnMenuText
_VnButtonShadow	_VnScrollBars
_VnButtonText	_VnTitleBarText
_VnDesktop	_VnWindowBackground
_VnGrayText	_VnWindowFrame
_VnHighlight	_VnWindowText
_VnHighlightText	

7.1.10 Standard Property Values

Empty object and null string values as as follows:

_VnNothing$	_VnNull$

Boolean values are as follows ("True" and "False" also work in VB 4.0):

_VnTrue	_VnFalse

Alignment options are as follows:

_VnCenter	_VnLeftJustify
_VnRightJustify	

Border styles are as follows:

_VnFixedDouble	_VnNone
_VnFixedSingle	_VnSizable

Mouse pointer types are as follows:

`_VnArrow`	`_VnSizeNeSw`
`_VnCrosshair`	`_VnSizeNS`
`_VnDefault`	`_VnSizeNwSe`
`_VnHourglass`	`_VnSizePointer`
`_VnIBeam`	`_VnSizeWE`
`_VnIconPointer`	`_VnUpArrow`
`_VnNoDrop`	

Window states are as follows:

`_VnMaximized`	`_VnMinimized`
`_VnNormal`	

Check box states are as follows:

`_VnChecked`	`_VnGrayed`
`_VnUnchecked`	

7.1.11 Common Dialog Flags

Constants for the Flags property for the ***ShowOpen*** and ***ShowSave*** common dialogs are:

`_cdlOFNAllowMultiselect`	`_cdlOFNNoDereferenceLinks`
`_cdlOFNCreatePrompt`	`_cdlOFNNoReadOnlyReturn`
`_cdlOFNExplorer`	`_cdlOFNNoValidate`
`_cdlOFNExtensionDifferent`	`_cdlOFNOverwritePrompt`
`_cdlOFNFileMustExist`	`_cdlOFNPathMustExist`
`_cdlOFNHelpButton`	`_cdlOFNReadOnly`
`_cdlOFNHideReadOnly`	`_cdlOFNShareAware`
`_cdlOFNNoChangeDir`	`_cdOFNLongNames`

Flags property constants for the ***ShowFont*** common dialog are:

`_cdlCFANSIOnly`	`_cdlCFNoVectorFonts`
`_cdlCFBoth`	`_cdlCFPrinterFonts`
`_cdlCFEffects`	`_cdlCFScalableOnly`
`_cdlCFFixedPitchOnly`	`_cdlCFScreenFonts`
`_cdlCFForceFontExist`	`_cdlCFHelpButton`
`_cdlCFLimitSize`	`_cdlCFTTOnly`
`_cdlCFNoSimulations`	`_cdlCFWYSIWYG`

Flags property constants for the ***ShowColor*** common dialog are:

`_cdCClFullOpen`	`_cdlCCRGBInit`
`_cdlCCPreventFullOpen`	`_cdlCCShowHelp`

Flags property constants for the ***ShowPrinter*** common dialog are:

`_cdlPDAllPages`	`_cdlPDPageNums`
`_cdlPDCollate`	`_cdlPDPrintSetup`
`_cdlPDDisablePrintToFile`	`_cdlPDPrintToFile`
`_cdlPDHidePrintToFile`	`_cdlPDReturnDC`
`_cdlPDNoPageNums`	`_cdlPDReturnIC`
`_cdlPDNoSelection`	`_cdlPDReturnDefault`
`_cdlPDNoWarning`	`_cdlPDSelection`
`_cdlPDHelpButton`	`_cdlPDUseDevModeCopies`

Constants for the HelpCommand property for the ***ShowHelp*** common dialog are:

`_cdlHelpCommand`	`_cdlHelpIndex`
`_cdlHelpContents`	`_cdlHelpKey`
`_cdlHelpContext`	`_cdlHelpPartialKey`
`_cdlHelpContextPopup`	`_cdlHelpQuit`
`_cdlHelpForceFile`	`_cdlHelpSetContents`
`_cdlHelpHelpOnHelp`	`_cdlHelpSetIndex`

7.2 Records

The record to hold change-list entries is:

```
RECORD VnChangedList
  FIELD Flag$1
  FIELD App$8
  FIELD Form$40
  FIELD Control$40
  FIELD ChgValue$40
  FIELD ChgValWhole$1
END RECORD VnChangedList
```

The record to hold a list of the controls created, used by `'VnCreateCtrls`, is:

```
RECORD VnControlsCreated
  FIELD CreatedCtrlName$40
END RECORD VnControlsCreated
```

The record to hold a list of information about VB control properties, used by `'VnGetPropInfoList`, is:

```
RECORD VnProperties
  FIELD Name$20
  FIELD DataType$1
  FIELD RunTimeAccess$1
  FIELD IsArray$1
  FIELD  Default$20
  FIELD PropValue$40
END RECORD VnProperties
```

The record to hold a list of controls for a given form, used by `'VnGetFormCtrlList`, is:

```
RECORD VnFormControls
  FIELD FormCtrlName$40
  FIELD FormCtrlTabIndex$1
  FIELD FormCtrlVisible$1
  FIELD FormCtrlType$3
  FIELD FormCtrlProp$80
END RECORD VnFormControls
```

The record to hold a list of loaded forms, used by `'VnGetLoadedFormList`, is:

```
RECORD VnLoadedForms
  FIELD LoadedFormName$40
  FIELD LoadedFormCaption$40
  FIELD LoadedFormVisible$1
END RECORD VnLoadedForms
```

The record to hold a list of printers, used by `'VnGetPrinterList`, is:

```
RECORD VnPrinters
  FIELD PrinterDeviceName$40
  FIELD PrinterDriverName$8
  FIELD PrinterPort$5
END RECORD VnPrinters
```

The record to hold a generic collection, used by `'VnGetCollectionList`, is:

```
RECORD VnCollection
  FIELD CollectionProp$60
END RECORD VnCollection
```

7.3 Variables

Variables for handling errors are:

```
VnError
VnErrMethod
VnErrMsg$80
```

Variables for returning the change list are:

```
VnChgNo
VnChgList$(0)# RECORDLENGTH(VnChangedList)
```

Variables for reporting the controls created are:

```
VnNumCtrlsCreated
VnCtrlsCreated$(0)# RECORDLENGTH(VnControlsCreated)
```

Variables for reporting properties for a given control are:

```
VnNumProps
VnPropInfo$(0)# RECORDLENGTH(VnProperties)
```

Variables for reporting controls for a given form are:

```
VnNumFormCtrls
VnFormCtrl$(0)# RECORDLENGTH(VnFormControls)
```

Variables for reporting loaded forms are:

```
VnNumLoadedForms
VnLoadedForm$(0)# RECORDLENGTH(VnLoadedForms)
```

Variables for reporting printers are:

```
VnNumPrinters
VnPrinter$(0)# RECORDLENGTH(VnPrinters)
```

Variables for reporting collections are:

```
VnNumMembers
VnMembeVnSysColorIndexRECORDLENGTH(VnCollection)
```

Variables for handling colors are:

`VnBlueVal`	`VnRGBColor(16)`
`VnColor$16`	`VnRGBColorIndex`
`VnColorSrc`	`VnSysColor$(19)16`
`VnGreenVal`	`VnSysColor(19)`
`VnRedVal`	`VnSysColorIndex`
`VnRGBColor$(16)13`	

Variables for reporting control value changes are:

`VnKey$1`	`VnKeyType$1`
`VnKeyin$5`	`VnKeyWin$1`
`VnKeyinMode$1`	`VnShift`
`VnKeyTran$5="True"`	

The variable for handling focus is:

`VnVbHasControl$1`

This variable used by `'VnPrintTo` should have room for an `INPUT SCREEN` of 24 rows by 80 columns (= 3248 for 24 rows by 132 columns):

`VnPrint$2080`

7.4 Subroutines

The following table lists the NPL subroutines grouped according to the type of operation they perform.

Operation	Subroutine	
Application Connections	`'VnClose`	`'VnOpen`
	`'VnCloseAll`	`'VnSetAppNum`
	`'VnGetAppNum`	
Management of the NPL Window	`'VnGetNplWndPos`	`'VnSetNplWndPos`
	`'VnGetNplWndShow`	`'VnSetNplWndShow`
	`'VnGetNplWndSize`	`'VnSetNplWndSize`
	`'VnGetNplWndTitle$`	`'VnSetNplWndTitle`
Property and Method Access	`'VnAddItems`	`'VnMethod`
	`'VnGetAlf$`	`'VnSetAlf`
	`'VnGetNum`	`'VnSetNum`
Record Access	`'VnGetRec$`	`'VnSetRec`
	`'VnGetRecSubset`	
Object Access	`'VnFreeObj`	`'VnIsObj$`
	`'VnGetObj$`	`'VnSetObj`
Font Translation	`'VnGetTran$`	`'VnSetTran`
Change-List Access	`'VnClearChgList`	`'VnGetChgList`

Color Manipulation	'VnGetColor 'VnSetRGB	'VnSetSysColor
Message Boxes and Input Boxes	'VnInpBox$	'VnMsgBox
Creating Controls “On The Fly”	'VnAt$ 'VnCreateCtrls 'VnDestroyCtrl 'VnInputScreen 'VnPrintAt	'VnPrintBox 'VnPrintCtrl 'VnPrintTo 'VnSetRowsCols
VB List Access	'VnGetCollectionList 'VnGetFormCtrlList 'VnGetLoadedFormList	'VnGetPrinterList 'VnGetPropInfoList
Event-Driven Support	'VnSleep	'VnWakeup
Error Handling	'VnErrFunc 'VnErrNum	'VnGetVbError
Miscellaneous	'VnCenter 'VnCmd 'VnConvNum$ 'VnDetect	'VnGetVer$ 'VnObj$ 'VnObj3$

Table 7.7 NPL Subroutines by Type

The remainder of this chapter contains detailed discussions of each of these routines.

7.4.1 'VnAddItems

Syntax

PROCEDURE 'VnAddItems (
    /POINTER *_Object$*,
    /POINTER *Items $()* **)**

Parameters

_Object$
List box, combo box, or grid to which to add the items

Items$()
Items to be added to *_Object$*

Description

This procedure adds an array of items to a list box, combo box or grid by calling the VB **AddItem** method of the object for each item in the array. `'VnMethod` is used to call the **AddItem** method, so none of the array elements can contain the global constant `_VnDelim$` , which is used to separate parameters when calling methods.
The items are appended to the list maintained by the object unless the object specifies otherwise. For example, if the Sorted property of a list box or combo box is True then the items will be inserted into the list at the appropriate position.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnAddItems("MyForm.Colors", VnRGBColor$())
```

See Also

'VnMethod

7.4.2 'VnAt$

Syntax

FUNCTION 'VnAt$ (
  /POINTER *_Row*,
  /POINTER *_Col* **)**

Parameters

_Row
  Row number.

_Col
  Column number.

Description

This function emulates the NPLAT(x,y) function when embedded in a PRINT statement. It returns characters which, when embedded in the VnPrint$ buffer, cause explicit positioning of the following text when VnPrint$ is subsequently sent to VB using the 'VnPrintTo function.

Return Values

Value	**Meaning**
VnError	Returns 0 if successful, otherwise, returns a >0 value

Example

```
PRINT TO VnPrint$;'VnAt$(2,4);"Name:"
PRINT TO VnPrint$;'VnAt$(4,4);"Address:"
PRINT TO VnPrint$;'VnAt$(10,4);"Phone:"
'VnPrintTo("MyForm")
```

See Also

'VnPrintAt
'VnPrintTo
'VnSetRowsCols

7.4.3 'VnCenter

Syntax

PROCEDURE 'VnCenter (
    /POINTER *_Object$* **)**

Parameters

_Object$
    The form object to be centered.

Description

This procedure centers a form on the screen. It does this by setting the Top and Left properties of the form based on the current sizes of the screen and the form.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnCenter("MainForm")
```

See Also

VB procedure**VnCenter**

7.4.4 'VnClearChgList

Syntax **PROCEDURE 'VnClearChgList**

Description This procedure clears the current change list, effectively erasing all changes that have been recorded so far.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnClearChgList
```

See Also **'VnCmd**
'VnGetChgList

7.4.5 'VnClose

Syntax

PROCEDURE 'VnClose

Description

This procedure closes a connection between an NPL program and a VB program as created by `'VnOpen`. The current application number is used to determine which connection is being closed. This value can be retrieved and/or set by the `'VnGetAppNum` and `'VnSetAppNum` procedures, respectively.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnClose
```

See Also

'VnCloseAll
'VnGetAppNum
'VnOpen
'VnSetAppNum

7.4.6 'VnCloseAll

Syntax

PROCEDURE 'VnCloseAll

Description

This procedure closes all connections between an NPL program and one or more VB programs as created by`'VnOpen` .

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnCloseAll
```

See Also

'VnClose
'VnGetAppNum
'VnOpen
'VnSetAppNum

7.4.7 'VnCmd

Syntax

PROCEDURE 'VnCmd (
  /POINTER *_Object$*,
  Cmd$20,
  /POINTER *_Value$* **)**

Parameters

_Object$
  The object on which the command will be performed.

Cmd$
  The command to be performed.

_Value$
  Command-specific parameters.

Description

This procedure has three purposes:

- It is the central routine used in change-list programming.
- It is used to issue some special commands to VB that can't be performed by setting properties or calling methods.
- It is used to add developer-defined commands to perform specialized tasks.

For more information, see "Remarks."

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

`VnChgNo` and `VnChgList$()` will be set for the VB **Show**, **Show Modeless**, and **Set Focus** commands.

Remarks

In the case of the change-list commands (`Show, Show Modeless`, and `Set Focus`) the procedure will only return when a Hot control has been triggered in the VB program. In this case, the `VnChgNo` variable will indicate the number of entries in the `VnChgList$()` array. In all other cases, the procedure returns immediately and `VnChgNo` and `VnChgList$()` are not affected.

Change-List Commands	Description
`Show`	Shows a form modally (the user can only access the form being shown) and clears the change list. The focus is set to the first control in the tab order. The `_Value$` parameter is not used for this command.
`Show Modeless`	Shows a form modelessly (the user can still access any form on the screen) and clears the change list. The focus is set to the first control in the tab order. The `_Value$` parameter is not used for this command.
`Set Focus`	Shows a form modelessly (the user can still access any form on the screen) and clears the change list. The focus is set to the first control in the tab order if just a form name is passed in `_Object$` or to the specified control if a form name and a control name are passed in `_Object$`. The `_Value$` parameter is not used for this command
`Resume`	Returns control to the form and the control that currently have the focus without clearing the change list. The `_Object$` and `_Value$` parameters are not used for this command

Special VB Commands	Description
`Load`	Loads a form into memory without showing it. This command can be used when you want to preload a form at program startup. The form can then be displayed or hidden when it is needed without incurring the time penalty of loading the form each time. In practice, this command is seldom used because the load time for most forms is negligible. Also, because any reference to a form (that is, setting and getting

	properties, calling methods, and so on) causes the form to be loaded, it is recommended that this command be avoided. The _Value$ parameter isn’t used for this command.
Unload	Closes a form and removes it from memory. It may not always be necessary to unload your forms using this command as some user interface actions will automatically unload a form. For example, selecting the ***Close*** item on the system menu of a form will unload the form. On the other hand, clicking the ***OK*** button of a form will do nothing unless your program tells it to. The _Value$ parameter is not used for this command.
Clear Form	Clears a form’s control values and unloads all control array members except the base control (index number 0). This command is used when generating screens “on the fly”. The _Value$ parameter isn’t used for this command.
Set Props	Sets the values of one or more of an object’s properties. The _Value$ parameter contains a list of property “name=value” pairs separated by the delimiter character _VnDelim$, the default for which is the “pipe” symbol (|)..
Load Picture	Loads a graphics file into a Picture, Icon, or DragIcon property. The _Value$ parameter is the name of the graphics file. This command calls the VB **LoadPicture** function. Most graphics properties are set at design time. Use this function only when graphics need to be

changed at run time.

Obsolete Commands (from Visual NPL 1.0)	Description
Display	Makes a form visible by setting its Visible property to True. Set the property directly rather than using this command.
Hide	Makes a form invisible by setting its Visible property to False. Set the property directly rather than using this command.
Add Item	Adds an item to a list box or combo box. Call the object’s **AddItem** method with **VnMethod** rather than using this command.
Remove Item	Removes an item from a list box or combo box. Call the object’s **RemoveItem** method with **VnMethod** rather than using this command.
Clear	Removes all items from a list box or combo box. Call the object’s **Clear** method with **VnMethod** rather than using this command.

Any command other than the those listed in this section is considered to be a developer-defined command and is passed on to the **VnDevDef** function in VB.

Examples

```
'VnCmd("MainForm","Show"," ")
'VnCmd("MainForm","Show Modeless"," ")
'VnCmd("MainForm.InputField","Set Focus"," ")
'VnCmd(" ","Resume"," ")
'VnCmd("MainForm","Load"," ")
'VnCmd("MainForm","Unload"," ")
'VnCmd("MainForm","Clear Form"," ")
'VnCmd("MainForm","Set Props","Top=100|Left=100")
'VnCmd("MainForm.Icon","Load Picture","SWOOSH.ICO")
```

See Also **'VnCallMethod**

7.4.8 'VnConvNum$

Syntax

FUNCTION 'VnConvNum$ (
  /POINTER *_Value* **)**

Parameters

_Value
  Numeric value to be converted.

Description

This procedure converts a numeric value into a string with no leading or trailing blanks.

Return Value

Returns the numeric value as a string.

Example

```
PRINT 'VnConvNum$(32);'VnConvNum$(76)

3276
```

7.4.9 'VnCreateCtrls

Syntax

PROCEDURE 'VnCreateCtrls (

/POINTER *_Form$*,

/POINTER *_Control$*,

/POINTER *_NumCtrls* **)**

Parameters

_Form$
Form on which to create the controls.

_Control$
Name of the control array to use.

_NumCtrls
Number of controls to create.

Description

Although VB is especially useful for creating forms and controls at design time, it can also create these objects at run time. This is called creating *controls "on the fly*." The `'VnCreateCtrls` procedure creates one or more controls by duplicating a *base control*. The new controls copy all of the properties of the base control except Index and TabIndex, which are both set to the next available value by VB. Setting properties of the base control prior to the `'VnCreateCtrls` call permits those properties to be replicated in the new controls. After a control is created, it can be displayed at a particular position on the form by calling the `'VnPrintCtrl` procedure (or by using `'VnSetAlf` and `'VnSetNum`).

The maximum number of elements per control array is 255.

Note Although the Visual NPL 1.0 `'VnCreateCtrl` procedure created a single control at a time, the Visual NPL 2.0 `'VnCreateCtrls` procedure creates multiple controls. Notice the difference in the procedure names.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
; create 3 label controls
'VnCreateCtrls("VnplChar","Label",2)
'VnPrintCtrl("Frm.Label(1)",2,2,1,1,"Name:"," ")
'VnPrintCtrl("Frm.Label(2)",4,2,1,1,"Address:"," ")
'VnPrintCtrl("Frm.Label(3)",6,2,1,1,"Phone:"," ")
```

See Also

'VnDestroyCtrl
'VnPrintAt
'VnPrintBox
'VnPrintCtrl
'VnPrintTo

7.4.10 'VnDestroyCtrl

Syntax

PROCEDURE 'VnDestroyCtrl (
/POINTER *_Object$* **)**

Parameters

_Object$
A control created by `'VnCreateCtrls` .

Description

This procedure is used to destroy the controls created by `'VnCreateCtrls` . The control should always be a member of an array and you should never destroy element 0 of the array. `'VnDestroyCtrl` is called implicitly when the form is unloaded.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnDestroyCtrl("Frm.Label(1)")
'VnDestroyCtrl("Frm.Label(2)")
```

See Also

'VnCreateCtrls
'VnPrintAt
'VnPrintBox
'VnPrintCtrl
'VnPrintTo

7.4.11 'VnDetect

Syntax

DEFFN 'VnDetect

Description

This `DEFFN` is used to detect whether or not the NPL library (**`VNPL16.DLL`**) is loaded. Calling `'VnDetect` and handling the error condition is used to determine whether the NPL library is present or not. If it doesn’t exist, then modules requiring the library are not loaded.
`'VnDetect` can also be called by its number, which is 32116.

Return Value

None.

Example

```
GOSUB 'VnDetect
ERROR GOSUB 'NoLibrary
```

7.4.12 'VnErrFunc

Syntax **PROCEDURE 'VnErrFunc**

Description This procedure puts the error message corresponding to the current error number (`VnError`) into `VnErrMsg$`. If the current error handling method (`VnErrMethod`) is set to `_VnErrCallFunc`, the error message will also be displayed using the `'VnMsgBox` function.
This is a developer-modifiable routine that exists in the `"VnplDev"` module. To change the text of the error messages simply change the text that appears within this function. This can be useful for handling foreign-language issues.
This procedure is called by almost all of the other NPL routines whenever an error occurs. It should not be called directly from your NPL program.

Return Values None

See Also **'VnErrNum**

7.4.13 'VnErrNum

Syntax

FUNCTION 'VnErrNum

Description

This function returns an NPL error code based on the current error number (`VnError`). If the current error handling method (`VnErrMethod`) is set to `_VnErrSignalError`, this function is called whenever an error occurs in one of the other NPL routines as follows:

```
RETURN ERROR ('VnErrNum)
```

By default, this function returns an error code in the range 601-627. You can change this function if the range conflicts with error codes already in use by your program.
This procedure is called by almost all of the other NPL routines whenever an error occurs. It should not be called directly from your NPL program.

Return Values

The NPL error code corresponding to the current value of `VnError`.

See Also

'VnErrFunc

7.4.14 'VnFreeObj

Syntax

PROCEDURE 'VnFreeObj (
/POINTER *Object$* **)**

Parameters

Object$
Object reference to be freed.

Description

This procedure frees an object reference created by `'VnGetObj$` .

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Obj$_VnObjLen
;
Obj$='VnGetObj$("MainForm.OKButton")
;
PRINT 'VnGetNum('VnObj$(Obj$,"Visible"))
;
'VnFreeObj(Obj$)
```

See Also

'VnGetObj$
'VnIsObj
'VnSetObj

7.4.15 'VnGetAlf$

Syntax

FUNCTION 'VnGetAlf$ (
    /POINTER *_Object$* **)**

Parameters

_Object$
    The property to get.

Description

This function returns the value of a property as a string. If it's not a string property, it will be converted to one.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

This function also returns the property value as a string.

Example

```
PRINT 'VnGetAlf$("MainForm.Caption")
```

See Also

'VnGetNum
'VnGetObj$
'VnSetAlf
'VnSetNum
'VnSetObj

7.4.16 'VnGetAppNum

Syntax

FUNCTION 'VnGetAppNum

Description

This function returns the *current application number* as set by the most recent call to `'VnOpen` or `'VnSetAppNum` . If there is no current application the function returns -1. This function is intended for use with NPL programs that connect to multiple VB programs.

Return Values

This function returns the *current application number.*

Example

```
AppNum='VnGetAppNum
```

See Also

'VnClose
'VnCloseAll
'VnOpen
'VnSetAppNum

7.4.17 'VnGetChgList

Syntax

PROCEDURE 'VnGetChgList

Description

This procedure retrieves the current change list. `VnChgNo` and `VnChgList$()` will be set appropriately.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnGetChgList
```

See Also

'VnCmd
'VnClearChgList

7.4.18 'VnGetCollectionList

Syntax

PROCEDURE 'VnGetCollectionList (
/POINTER *_Object$*,
/POINTER *_PropName$* **)**

Parameters

Object$
Collection from which to get property values.

_PropName$
Property of which to get the values.

Description

This procedure gets the value of a particular property for each member of a collection. The number of elements is returned in the `VnNumMembers` global variable and the values are returned in the `VnMember$()` global variable. Each element of `VnMember$()` is a `VnCollection` record, which consists of one string field named `CollectionProp$`. This field will hold the property value for each member.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value
`VnMember$()`	Property values for each member
`VnNumMembers`	Number of members in the collection

Example

```
DIM I
;
'VnGetCollectionList("Forms","Name")
;
PRINT "Names of loaded forms:"
FOR I=1 TO VnNumMembers
  PRINT VnMember$(I) .CollectionProp$
NEXT I
```

See Also

'VnGetFormCtrlList
'VnGetLoadedFormList
'VnGetPrinterList
'VnGetPropInfoList

7.4.19 'VnGetColor

Syntax

PROCEDURE 'VnGetColor (

/POINTER *_Color*)

Parameters

_Color

A color value as used by VB.

Description

This procedure breaks a color value into its component parts. The global variable `VnColorSrc` is set as follows to indicate the type of color value:

`_VnRGBColor`	RGB color
`_VnSysColor`	System color
0	Neither an RGB nor a system color

If the type is RGB color, then the `VnRedVal`, `VnGreenVal`, and `VnBlueVal` global variables will be set to the component color parts. If the color is one of the standard RGB colors, then the `VnRGBColorIndex` global variable will be set to the index of the color value in the `VnRGBColor` array and the `VnColor$` global variable will be set to the name of the color.
If the type is a system color, then the `VnSysColorIndex` global variable will be set to the index of the color value in the `VnSysColor` array. The `VnColor$` global variable will be set to the name of the color.

If it's neither an RGB color nor a system color, then no other global variables are set.

Return Values

Value	Meaning
`VnColorSrc`	The type of color, RGB, system, or other
`VnRedVal`	Red component for RGB colors
`VnGreenVal`	Green component for RGB colors
`VnBlueVal`	Blue component for RGB colors
`VnRGBColorIndex`	Index into `VnRGBColor` for RGB colors
`VnSysColorIndex`	Index into `VnSysColor` for system colors
`VnColor$`	Name of the color for RGB and system colors

Example

```
DIM Color
;
; get the background color for the main form
; and break it into its component parts
'VnGetColor('VnGetNum("MainForm.BackColor"))
;
; print the color source value
PRINT "Color Source: " ;VnColorSrc
;
; print the color information
SWITCH VnColorSrc
  ;
  CASE 0
    PRINT "Unknown color"
  ;
  CASE _VnRGBColor
    PRINT "RGB Color: " ;
    IF VnRGBColorIndex<>0
      PRINT VnColor$
    ELSE
      PRINT VnRedVal ;VnGreenVal;VnBlueVal
    END IF
  ;
  CASE _VnSysColor
    PRINT "System Color:" ;VnColor$
  ;
END SWITCH
```

See Also

'VnSetRGB
'VbSetSysColor

7.4.20 'VnGetFormCtrlList

Syntax

PROCEDURE 'VnGetFormCtrlList (
    /POINTER *_Object$*,
    /POINTER *_PropName$* **)**

Parameters

_Object$
Form for which to get controls.

_PropName$
Control property for which to get values.

Description

This procedure gets this information for each control on a form:

- The name of the control
- The tab index of the control
- Whether or not the control is visible
- The type of control
- The value of a specific property

The number of controls is returned in the `VnNumFormCtrls` global variable and the control information is returned in the `VnFormCtrl$()` global variable. Each element of `VnFormCtrl$()` is a `VnFormControls` record, which consistsof the following fields:

```
FormCtrlName$
FormCtrlTabIndex$
FormCtrlVisible$
FormCtrlType$
FormCtrlProp$
```

These fields correspond one-for-one with the preceding list.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM I
;
'VnGetFormCtrlList("MainForm","Top")
;
PRINT "Control Tops:"
FOR I=1 TO VnNumFormCtrls
  PRINT VnFormCtrl$(I) .FormCtrlName$;
  PRINT VnFormCtrl$(I) .FormCtrlProp$
NEXT I
```

See Also

'VnGetCollectionList
'VnGetLoadedFormList
'VnGetPrinterList
'VnGetPropInfoList

7.4.21 'VnGetLoadedFormList

Syntax **PROCEDURE 'VnGetLoadedFormList**

Description

This procedure gets the following information for each form that is currently loaded:

- The name of the form
- The form's caption
- Whether or not the form is visible

The number of forms is returned in the `VnNumLoadedForms` global variable and the form information is returned in the `VnLoadedForm$()` global variable. Each element of `VnLoadedForm$()` is a `VnLoadedForm` record, which consistsof the following fields:

```
LoadedFormName$
LoadedFormCaption$
LoadedFormVisible$
```

These fields correspond one-for-one with the preceding list.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM I
;
'VnGetLoadedFormList
;
PRINT "Loaded forms:"
FOR I=1 TO VnNumLoadedForms
  PRINT VnLoadedForm$(I) .LoadedFormName$;" ";
  PRINT VnLoadedForm$(I) .LoadedFormCaption$;" ";
  PRINT VnLoadedForm$(I) .LoadedFormVisible$
NEXT I
```

See Also

'VnGetCollectionList
'VnGetFormCtrlList
'VnGetPrinterList
'VnGetPropInfoList

7.4.22 'VnGetNplWndPos

Syntax

PROCEDURE 'VnGetNplWndPos (
  /POINTER *Left*,
  /POINTER *Top* **)**

Parameters

Left
  Left coordinate of the NPL run-time window.

Top
  Top coordinate of the NPL run-time window.

Description

This procedure gets the position (upper left corner) of the NPL run-time window. All coordinates are in twips (1440 per inch).

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Left ,Top
;
'VnGetNplWndPos(Left,Top)
;
PRINT Left ,Top
```

See Also

'VnGetNplWndSize
'VnGetNplWndShow
'VnGetNplWndTitle$
'VnSetNplWndPos
'VnSetNplWndShow
'VnSetNplWndSize
'VnSetNplWndTitle

7.4.23 'VnGetNplWndSize

Syntax

PROCEDURE 'VnGetNplWndSize (
 /POINTER *Width*,
 /POINTER *Height* **)**

Parameters

Width
 Width of the NPL run-time window.

Height
 Height of the NPL run-time window.

Description

This procedure gets the size (width and height) of the NPL run-time window. All sizes are in twips (1,440 per inch).

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Width ,Height
;
'VnGetNplWndSize(Width,Height)
;
PRINT Width ,Height
```

See Also

'VnGetNplWndPos
'VnGetNplWndShow
'VnGetNplWndTitle$
'VnSetNplWndPos
'VnSetNplWndShow
'VnSetNplWndSize
'VnSetNplWndTitle

7.4.24 'VnGetNplWndShow

Syntax

FUNCTION 'VnGetNplWndShow

Description

This procedure determines whether or not the NPL run-time window is visible.

Return Values

Value	Meaning
`_VnShow`	Returned if the NPL run-time window is visible
`_VnHide`	Returned if the NPL run-time window is not visible

Example

```
DIM Ret
;
IF 'VnGetNplWndShow=_VnShow
  Ret='VnMsgBox("Visible","Runtime Window",0)
ELSE
  Ret='VnMsgBox("Invisible","Runtime Window",0)
END IF
```

See Also

'VnGetNplWndPos
'VnGetNplWndSize
'VnGetNplWndTitle$
'VnSetNplWndPos
'VnSetNplWndShow
'VnSetNplWndSize
'VnSetNplWndTitle

7.4.25 'VnGetNplWndTitle$

Syntax

FUNCTION 'VnGetNplWndTitle$

Parameters

This function takes as a parameter the title of the NPL run-time window.

Description

This procedure gets the title (caption) of the NPL run-time window.

Return Values

Value	Meaning
VnError	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Title$100
;
'VnGetNplWndTitle$(Title$)
;
PRINT Title$
```

See Also

'VnGetNplWndPos
'VnGetNplWndShow
'VnGetNplWndSize
'VnSetNplWndPos
'VnSetNplWndShow
'VnSetNplWndSize
'VnSetNplWndTitle

7.4.26 'VnGetNum

Syntax

FUNCTION 'VnGetNum (
    /POINTER *_Object$* **)**

Parameters

_Object$
    The property to get.

Description

This function returns the value of a property as a number. If it's not a numeric property, it will be converted to one if possible.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

This function also returns the value of the property as a number.

Example

```
PRINT 'VnGetNum("MainForm.Top")
```

See Also

'VnGetAlf$
'VnGetObj$
'VnSetAlf
'VnSetNum
'VnSetObj

7.4.27 'VnGetObj$

Syntax

FUNCTION 'VnGetObj$ (
/POINTER *_Object$* **)**

Parameters

_Object$
Object for which to get a reference.

Description

This function returns an object reference for a particular object. When you are done with the object, you must call`'VnFreeObj` to free the reference.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

This function also returns the object reference for the object.

Example

```
DIM Obj$_VnObjLen
;
Obj$='VnGetObj$("MainForm.OKButton")
;
PRINT 'VnGetNum('VnObj$(Obj$,"Visible"))
;
'VnFreeObj(Obj$)
```

See Also

'VnFreeObj
'VnIsObj
'VnSetObj

7.4.28 'VnGetPrinterList

Syntax

PROCEDURE 'VnGetPrinterList

Description

This procedure gets the following information for each printer in the Printers collection:

- The name of the printer device
- The name of the printer driver
- The printer port

The number of printers is returned in the `VnNumPrinters` global variable and the printer information is returned in the `VnPrinter$()` global variable. Each element of `VnPrinter$()` is a `VnPrinters` record, which consists of the following fields:

```
PrinterDeviceName$
PrinterDriverName$
PrinterPort$
```

These fields correspond one-for-one with the preceding list.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM I
;
'VnGetPrinterList
;
PRINT "Printers:"
FOR I=1 TO VnNumPrinters
  PRINT VnPrinter$(I) .PrinterDeviceName$;" ";
  PRINT VnPrinter$(I) .PrinterDriverName$;" ";
  PRINT VnPrinter$(I) .PrinterPort$
NEXT I
```

See Also

'VnGetCollectionList
'VnGetFormCtrlList
'VnGetLoadedFormList
'VnGetPropInfoList

7.4.29 'VnGetPropInfoList

Syntax

PROCEDURE 'VnGetPropInfoList (
/POINTER *_Object$* **)**

Parameters

_Object$
Object for which to get properties.

Description

This procedure gets the following information for each property of a particular object:

- The name of the property
- The data type of the property
- How the property can be accessed at run time
- Whether or not the property is an array
- The property's current value

The number of properties is returned in the `VnNumProps` global variable and the property information is returned in the `VnPropInfo$()` global variable. Each element of `VnPropInfo$()` is a `VnProperties` record, which consists of the following fields:

```
Name$
DataType$
RunTimeAccess$
IsArray$
PropValue$
```

These fields correspond one-for-one with the preceding list.

Note The `VnProperties` record also contains a field named `Default$`, which was used in Visual NPL 1.0 to retrieve the property's default value. This is no longer supported and this field will always be blank.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM I
;
'VnGetPropInfoList("MainForm")
;
PRINT "MainForm properties:"
FOR I=1 TO VnNumProps
  PRINT VnPropInfo$(I) .Name$;" ";
  PRINT VnPropInfo$(I) .DataType$;" ";
  PRINT VnPropInfo$(I) .PropValue$
NEXT I
```

See Also

'VnGetCollectionList
'VnGetFormCtrlList
'VnGetLoadedFormList
'VnGetPrinterList

7.4.30 'VnGetRec$

Syntax

FUNCTION 'VnGetRec$ (
/POINTER *_Object$*,
/POINTER *_RecName$* **)**

Parameters

_Object$
The form from which to get the record fields.

_RecName$
The name of the record to use when getting fields.

Description

This function returns all of the field values for a particular NPL record from the controls on a form. The order, placement, and data types for the values will be as declared in the `PUBLIC` record named in the `_RecName$` parameter.

To identify a control on the form as being a record field, put the record and field name (separated by a period and omitting any dollar signs) into the control's Tag property. For example, putting `CustRec.Address` into the Tag property of a TextBox control will cause the control's value to be retrieved into the `Address` field of a buffer formatted according to record `CustRec.`

Note When using this procedure, all fields defined in the NPL record must have a corresponding control on the VB form, or an error will be reported.

Return Values

This function returns the buffer containing the record values.

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM CustRec$1000
;
CustRec$= 'VnGetRec("MyForm","Customer")
```

See Also

'VnGetRecSubset
'VnSetRec

7.4.31 'VnGetRecSubset

Syntax

PROCEDURE 'VnGetRecSubset (
  /POINTER *_Object$*,
  /POINTER *_RecName$*,
  /POINTER *Buffer$* **)**

Parameters

_Object$
  Form from which to get the record fields.

_RecName$
  Name of the record to use when getting fields.

Buffer$
  Buffer into which to put the record values.

Description

This procedure is a variation on the`'VnGetRec$` function. Although the `'VnGetRec$` function requires all fields in the record to be defined in the Tag properties of the target form, the`'VnGetRecSubset` procedure does not.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM CustRec$1000
;
'VnGetRecSubset("MyForm","Customer",CustRec$)
```

See Also

'VnGetRec
'VnSetRec

7.4.32 'VnGetTran$

Syntax

FUNCTION 'VnGetTran$

Description

This function gets the current font translation table. The translation table consists of zero or more pairs of characters in which the first is the NPL character and the second is the VB character.

Return Values

This function returns the current font translation table.

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Tran$1000
;
Tran$='VnGetTran
```

See Also

'VnSetTran

7.4.33 'VnGetVbError

Syntax

FUNCTION 'VnGetVbError (
  /POINTER *ErrorMsg$* **)**

Parameters

ErrorMsg$
  The most recent VB error message.

Description

When one of the NPL routines generates an error, it could be due to an error in VB. If this is the case, then the error code `_VnErrVbError` will be returned. In this case, the `'VnGetVbError` function can be called to get the VB error information. The function returns the VB error code and sets the `ErrorMsg$` parameter to the VB error message.

Return Values

This function returns the most recent VB error code.

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Ret ,VbError$100
;
Ret=result of some Visual NPL operation
;
IF Ret=_VnErrVbError
  VbRet= 'VnGetVbError(VbError$)
  PRINT "VB error " ;VbRet;" - ";VbError$
END IF
```

7.4.34 'VnGetVer$

Syntax

FUNCTION 'VnGetVer$

Description

This function returns the version number of Visual NPL. This string is a three-part number of the format X.YY.ZZ where:

- X is the major version number
- YY is the minor version number
- ZZ is the subminor version number

A typical output result might be 2.00.04.

Return Values

This function returns the Visual NPL version number.

Example

```
PRINT "I'm using Visual NPL, Version ";'VnGetVer$
```

7.4.35 'VnInpBox$

Syntax

FUNCTION 'VnInpBox$ (
Title$80,
Prompt$80,
Flags,
/POINTER *_InValue$* **)**

Parameters

Title$
Title (caption) of the input box window

Prompt$
Prompt within the input box window

Flags
Indicates that the value is either a single-line or a multiline value

_InValue$
Initial value

Description

This function prompts the user for an input value, using a small form with a single or multiline TextBox control. If the user clicks the *OK* button, the value in the TextBox is returned. If the user clicks *Cancel*, an empty string is returned.
To create a single-line input box, set the `Flags` parameter to 0. To create a multiline input box, set the `Flags` parameter to `_VnMultiline`.

Return Values

This function returns the value entered or an empty string if *Cancel* is clicked.

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Answer$3
;
Answer$= 'VnInpBox("Delete Everything",
                  "Are you sure, Yes or No?  ”,
                  0," ")
;
IF Answer="Yes"
  ; delete everything
END IF
```

See Also

'VnMsgBox

7.4.36 'VnInputScreen

Syntax

PROCEDURE 'VnInputScreen (
 /POINTER *_Object$*,
 /POINTER *_Row*,
 /POINTER *_Col*,
 Height,
 Width **)**

Parameters

_Object$
 Object on which to draw the screen image

_Row
 Row on which to start capturing the screen

_Col
 Column on which to start capturing the screen

Height
 The number of rows to capture

Width
 The number of columns to capture

Description

This procedure emulates a combination of the NPL `INPUT SCREEN` and `PRINT SCREEN` commands, with the `INPUT SCREEN` being performed on the NPL window and the `PRINT SCREEN` being performed on the VB object. It could be used to pass the background of a NPL data-entry screen to VB. Only text is transferred, no attributes or colors. Because this procedure uses the VB **Print** method, the print data may be displayed to a form or a picture box control, to a printer, or to the VB Debug window.

Note The NPL window need not be visible to have data displayed and captured by `'VnInputScreen`.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
; clear NPL screen
PRINT HEX(03)
;
; print text on NPL screen (need not be visible)
PRINT AT(2,4);"Name:"
PRINT AT(4,4);"Address:"
PRINT AT(6,4);"Phone:"
;
; transfer NPL screen text to a VB form
'VnInputScreen("MyForm",0,0,10,40)
```

See Also

'VnPrintAt
'VnPrintBox
'VnPrintCtrl
'VnPrintTo
'VnSetRowsCols

7.4.37 'VnIsObj$

Syntax

FUNCTION 'VnIsObj$ (
/POINTER *_Object$* **)**

Parameters

_Object$
The string that may or may not be an object reference

Description

This procedure emulates a combination of the NPL `INPUT SCREEN` and `PRINT SCREEN` commands, with the `INPUT SCREEN` being performed on the NPL window and the `PRINT SCREEN` being performed on the VB object. It could be used to pass the background of a NPL data-entry screen to VB. Only text is transferred, no attributes or colors. Because this procedure uses the VB **Print** method, the print data may be displayed to a form or a picture box control, to a printer, or to the VB Debug window.

Note The NPL window need not be visible to have data displayed and captured by `'VnInputScreen` .

This function determines whether or not a string is an object reference. An object reference is a string that is:

- At least `_VnObjLen` characters long
- Starts with `"VnOb"`
- Ends with `"bOnV"`

The middle characters are the object reference; the wrapper is used to distinguish objects from normal strings. If the preceding conditions are met, then the function returns “Y,” otherwise, it returns “N.”

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
DIM Obj$_VnObjLen
;
PRINT 'VnIsObj$(Obj$);
;
Obj$='VnGetObj$("MainForm");
;
PRINT 'VnIsObj$(Obj$)
;
'VnFreeObj(Obj$)
```

See Also

'VnFreeObj
'VnGetObj

7.4.38 'VnMethod

Syntax

PROCEDURE 'VnMethod (
/POINTER *_Object$,*
/POINTER *_Parms$* **)**

Parameters

_Object$
Object and method to call.

_Parms$
Method parameters.

Description

This procedure calls a method for a VB object. The method parameters are passed as a list of items separated by the global constant `VnDelim$` , the default for which is the “pipe” symbol (|). There are actually three versions of this routine—the procedure listed in the preceding section, a function that returns a numeric, and a function that returns a string as follows:

```
PROCEDURE 'VnMethod(/POINTER  _Object$,/POINTER _Parms$)
FUNCTION 'VnMethod(/POINTER  _Object$,/POINTER _Parms$)
FUNCTION 'VnMethod$(/POINTER  _Object$,/POINTER _Parms$)
```

Although most methods are called as procedures that simply perform some operation on their object, some are used to do calculations and then return the results. The version you use depends on the method being called.

Note `'VnMethod` is currently limited to passing its parameters into the method. Any changes made to the parameters by the method will not be returned to your NPL program.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnMethod(“MainForm.ListBox.AddItem","One")
'VnMethod(“MainForm.Move","100|200")
N='VnMethod(“MainForm.TextWidth","Some text")
```

See Also

7.4.39 'VnMsgBox

Syntax

FUNCTION 'VnMsgBox (
  /POINTER *_Title$*,
  /POINTER *_Msg$*,
  Flags **)**

Parameters

_Title$
  Title (caption) of the message box window

_Msg$
  Message to appear within the message box window

Flags
  Icon and button mask

Description

This function shows a message in a small window. You use the *Flags* parameter to determine what type of icon (if any) to show beside the message as well as which buttons to use. The icons available are as follows:

```
_VnMbIconStop
_VnMbIconQuestion
_VnMbIconExc lamation
_VnMbIconInformation
```

The button combinations that are available are as follows:

```
_VnMbOk
_VnMbOkCancel
_VnMbAbortRetryIgnore
_VnMbYesNoCancel
_VnMbYesNo
_VnMbRetryCancel
```

The possible return codes are as follows:

```
_VnMbIdOk
_VnMbIdCancel
_VnMbIdAbort
_VnMbIdRetry
_VnIdIgnore
_VnIdYes
_VnIdNo
```

Return Values

This function returns an indication of which button was clicked.

Example

```
DIM Answer
;
Answer='VnMsgBox("Delete Everything","Are you sure?",
                 _VnMbYesNo+_VnMbIconQuestion)
;
IF Answer=_VnIdYes
  ; Delete everything here
END IF
```

See Also

'VnInpBox

7.4.40 'VnObj$

Syntax

FUNCTION 'VnObj$ (
  /POINTER *_Name1$*,
  /POINTER *_Name2$* **)**

Parameters

Name1$
  The first part of an object name

Name2$
  The second part of an object name

Description

This function builds an object name from two parts by concatenating the two, with the parts joined by a period. Neither part can be blank, as no checking is done to handle the case of blank names. This is done in order to make the routine as fast as possible.

Return Values

This function returns a combined object name.

Example

```
DIM Obj$_VnObjLen
;
Obj$='VnGetObj("MainForm")
;
PRINT 'VnGetAlf$('VnObj$(Obj$,"Name"))
;
'VnFreeObj(Obj$)
```

7.4.41 'VnObj3$

Syntax

FUNCTION 'VnObj3$ (
    /POINTER *_Name1$*,
    /POINTER *_Name2$*,
    /POINTER *_Name3$* **)**

Parameters

Name1$
    First part of an object name

Name2$
    Second part of an object name

Name3$
    Third part of an object name

Description

This function builds an object name from three parts by concatenating the three, with the parts joined by a period. None of the three parts can be blank, as no checking is done to handle the case of blank names. This is done in order to make the routine as fast as possible.

Return Values

This function returns a combined object name.

Example

```
DIM Form$20,Control$20,Property$20
;
Form$="SignOn"
Control$="Password"
Property$="Text"
;
PRINT 'VnGetAlf$('VnObj3$(Form$ ,Control$,Property$))
```

7.4.42 'VnOpen

Syntax

PROCEDURE 'VnOpen (
ExeName$260 **)**

Parameters

ExeName$
Name of the VB program to run

Description

This procedure opens a link with a VB program. The `ExeName$` parameter is the base name of the VB executable file (the file name without the full path and without the **`.EXE`** extension).
The procedure first looks for a connection control with this base name that is waiting to be connected to. If it can't find one, then it will look for the executable file in the standard Windows search path and run it. It will then look for the connection control again. If it still can't find it, or if it couldn't find or run the executable, it generates a Visual NPL error. Otherwise, the connection is made and your program continues on.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnOpen("BASENAME")
```

See Also

'VnClose
'VnCloseAll
'VnGetAppNum
'VnSetAppNum

7.4.43 'VnPrintAt

Syntax

PROCEDURE 'VnPrintAt (
  /POINTER *_Object$*,
  /POINTER *_Row*,
  /POINTER *_Col*,
  /POINTER *_Text$* **)**

Parameters

_Object$
  The form on which to print

_Row
  The row at which to print

_Col
  The column at which to print

_Text$
  The text to be printed

Description

This procedure emulates the NPL`PRINT AT` statement by printing a string at a specific location on a form.

Note The output from this procedure was buffered in Visual NPL 1.0; it is now sent to VB immediately.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnPrintAt("MyForm",2,2,"Name:")
'VnPrintAt("MyForm",4,2,"Address:")
'VnPrintAt("MyForm",6,2,"Phone:")
```

See Also

'VnPrintBox
'VnPrintCtrl
'VnPrintTo
'VnSetRowsCols

7.4.44 'VnPrintBox

Syntax

PROCEDURE 'VnPrintBox (
  /POINTER *_Object$*,
  /POINTER *_Row*,
  /POINTER *_Col*,
  /POINTER *_Height*,
  /POINTER *_Width*,
  /POINTER *_Color* **)**

Parameters

_Object$
  The form on which to print the box

_Row
  The left row at which to print the box

_Col
  The top column at which to print the box

_Height
  The number of rows in the box

_Width
  The number of columns in the box

_Color
  Background color of the box

Description

This procedure emulates the NPL `PRINT BOX` statement by drawing a box at a particular location on a form.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnPrintBox("MyForm",2,20,5,40,_VnLightGray)
```

See Also

'VnAt
'VnPrintAt
'VnPrintCtrl
'VnPrintTo
'VnSetRowsCols
'VnSetRGB
'VnSetSysColor

7.4.45 'VnPrintCtrl

Syntax

PROCEDURE 'VnPrintCtrl (
  /POINTER *_Object$*,
  /POINTER *_Row*,
  /POINTER *_Col*,
  /POINTER *_Height*,
  /POINTER *_Width*,
  /POINTER *_Value$*,
  /POINTER *_Buffer$* **)**

Parameters

_Object$
The form on which to print the control

_Row
The row at which to print the control

_Col
The column at which to print the control

_Height
The number of rows in the control

_Width
The number of columns in the control

_Value$
The initial value of the control

_Buffer$
Other property values

Description

This prints a control onto a form when generating *controls "on the fly."* The form being printed tois treated as a text-emulation window; its coordinates are expressed in characters rather than pixels. The control must have been created with `'VnCreateCtrl` . After creating the control, but before printing, you may change any of its properties.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnCreateCtrls("VnplChar","Label",2)
'VnPrintCtrl("Frm.Label(0)",2,2,1,1,"Name:"," ")
'VnPrintCtrl("Frm.Label(1)",4,2,1,1,"Address:"," ")
```

See Also

'VnAt
'VnPrintAt
'VnPrintBox
'VnPrintTo
'VnSetRowsCols

7.4.46 'VnPrintTo

Syntax

PROCEDURE 'VnPrintTo (
/POINTER *_Object$* **)**

Parameters

_Object$
Object to which to print

Description

This procedure emulates a subset of the NPL printing capability. Instead of using the NPL `PRINT` and `PRINTUSING` commands, use the NPL `PRINT TO` and `PRINTUSING TO` commands with the `VnPrint$` buffer. The `'VnPrintTo` procedure passes the print data accumulated in the `VnPrint$` buffer to VB to be displayed. Because this procedure uses the VB **Print** method, the print data may be displayed to a form or a picture box control, to a printer, or to the VB Debug window.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
PRINT TO VnPrint$;'VnAt$(2,4);"Name:"
PRINT TO VnPrint$;'VnAt$(4,4);"Address:"
PRINT TO VnPrint$;'VnAt$(10,4);"Phone:"
'VnPrintTo("MyForm")
```

See Also

'VnAt
'VnPrintAt
'VnPrintBox
'VnPrintCtrl
'VnSetRowsCols

7.4.47 'VnSetAlf

Syntax

PROCEDURE 'VnSetAlf (
/POINTER *_Object$*,
/POINTER *_Value$* **)**

Parameters

_Object$
The property to set

_Value$
The property value

Description

This function sets the value of a property from a string. If the property type is numeric (or something other than string), the string value will be converted to the appropriate type.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetAlf("MainForm.Caption","My Main Form")
```

See Also

'VnGetAlf$
'VnGetNum
'VnGetObj$
'VnSetNum
'VnSetObj

7.4.48 'VnSetAppNum

Syntax

PROCEDURE 'VnSetAppNum (
NewAppNum **)**

Parameters

NewAppNum
The new current application number

Description

This procedure sets the *current application number* to a new value. This value must have been returned by `'VnGetAppNum` after a call to `'VnOpen`. This procedure is intended for use with NPL programs that connect to multiple VB programs.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetAppNum(AppNum)
```

See Also

'VnClose
'VnCloseAll
'VnGetAppNum
'VnOpen

7.4.49 'VnSetNplWndPos

Syntax

PROCEDURE 'VnSetNplWndPos (
Left,
Top **)**

Parameters

Left
The new left coordinate of the NPL run-time window

Top
The new top coordinate of the NPL run-time window

Description

This procedure sets the position (upper, left corner) of the NPL run-time window. All coordinates are in twips (1440 per inch).

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNplWndPos(1440,2880)
```

See Also

'VnGetNplWndPos
'VnGetNplWndSize
'VnGetNplWndShow
'VnGetNplWndTitle$
'VnSetNplWndShow
'VnSetNplWndSize
'VnSetNplWndTitle

7.4.50 'VnSetNplWndShow

Syntax

PROCEDURE 'VnSetNplWndShow (
Mode **)**

Parameter

Mode
Indicates whether to show or hide the NPL run-time window

Description

This procedure shows or hides the NPL run-time window. If the `Mode` parameter is set to `_VnHide`, then the window is hidden. If the `Mode` parameter is set to `_VnShow`, then the window is shown.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNplWndShow(_VnHide)
'VnSetNplWndShow(_VnShow)
```

See Also

'VnGetNplWndPos
'VnGetNplWndSize
'VnGetNplWndShow
'VnGetNplWndTitle$
'VnSetNplWndPos
'VnSetNplWndSize
'VnSetNplWndTitle

7.4.51 'VnSetNplWndSize

Syntax

PROCEDURE 'VnSetNplWndSize (
Width,
Height **)**

Parameters

Width
The width of the NPL run-time window

Height
The height of the NPL run-time window

Description

This procedure sets the size (width and height) of the NPL run-time window. All sizes are in twips (1,440 per inch).

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNplWndSize(7200,7200)
```

See Also

'VnGetNplWndPos
'VnGetNplWndSize
'VnGetNplWndShow
'VnGetNplWndTitle$
'VnSetNplWndPos
'VnSetNplWndShow
'VnSetNplWndTitle

7.4.52 'VnSetNplWndTitle

Syntax

PROCEDURE 'VnSetNplWndTitle (
Title$80 **)**

Parameters

Title$
The new title (caption) for the NPL run time

Description

This procedure sets the title (caption) of the NPL run-time window.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNplWndTitle$("My Runtime Window")
```

See Also

'VnGetNplWndPos
'VnGetNplWndSize
'VnGetNplWndShow
'VnGetNplWndTitle
'VnSetNplWndPos
'VnSetNplWndShow
'VnSetNplWndSize

7.4.53 'VnSetNum

Syntax

PROCEDURE 'VnSetNum (
/POINTER *_Object$*,
/POINTER *_Value* **)**

Parameters

_Object$
The property to set

_Value
The property value

Description

This function sets the value of a property from a number. If the property type is nonnumeric, the number will be converted to the appropriate type.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNum("MainForm.Top",2880)
```

See Also

'VnGetAlf$
'VnGetNum
'VnGetObj$
'VnSetNum
'VnSetObj

7.4.54 'VnSetObj

Syntax

PROCEDURE 'VnSetObj (
/POINTER *_Object$*,
/POINTER *_Value$* **)**

Parameters

_Object$
The property to be set

_Value$
The object reference

Description

This procedure sets an “object” property to a new value. Note that the `_Object$` parameter must refer to some property of an object and not to the object itself.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetObj("Frm.Label(0).Container","Frm.Frame")
```

See Also

'VnFreeObj
'VnGetObj
'VnIsObj$

7.4.55 'VnSetRec

Syntax

PROCEDURE 'VnSetRec (
  /POINTER *_Object$*,
  /POINTER *_RecName$*,
  /POINTER *_Buffer$* **)**

Parameters

_Object$
  The form on which to set the controls

_RecName$
  The name of the record to use when setting fields

_Buffer$
  The record from which to set the controls

Description

This function sets values for a form's controls from field values for an NPL record. Order, placement, and data types for the values will be as declared in the public record named in the `_RecName$` parameter.

To identify a control on the form as a record field, put the record and field name (separated by a period and without dollar signs) into the control's Tag property. For example, putting `CustRec.Address` into the Tag property of a TextBox control will cause the control's value to be fetched into the `Address` field of a buffer formatted according to record `CustRec`.

Note When using this procedure, all fields defined in the NPL record must have a corresponding control on the VB form, or an error will be reported.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
RECORD /PUBLIC InfoRecord
   FIELD Name$30
   FIELD Address$40
   FIELD Phone$10
END RECORD
;
DIM Info$# RECORDLENGTH(InfoRecord)
;
'VnSetRec("InfoForm","InfoRecord",Info$)
```

See Also

'VnGetRec
'VnGetRecSubset

7.4.56 'VnSetRGB

Syntax

FUNCTION 'VnSetRGB (
/POINTER *_Red*,
/POINTER *_Green*,
/POINTER *_Blue* **)**

Parameters

_Red
Red color component

_Green
Green color component

_Blue
Blue color component

Description

This function creates an RGB color value from its red, green, and blue components. Each of the component colors is a number from 0 to 255 describing the intensity of that color in the combined color.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNum("MainForm.BackColor",'VnSetRGB(255,0,0))
```

See Also

'VnGetColor
'VnSetSysColor

7.4.57 'VnSetRowsCols

Syntax

PROCEDURE 'VnSetRowsCols (
/POINTER *_Object$*,
/POINTER *_NumRows*,
/POINTER *_NumCols* **)**

Parameters

_Object$
The form for which to set row and column mapping

_NumRows
The number of rows to use when mapping screen coordinates

_NumCols
The number of columns to use when mapping screen coordinates

Description

This function sets the number of rows and columns to be used when mapping NPL row and column numbers to the default twips coordinate system used by Visual Basic. In fact, this mapping will work regardless of which coordinate system VB is using (as set by the ScaleMode form property).

There are several important concerns regarding the use of this procedure:

- You should set the form width and height before calling this routine
- You must call this routine each time the user resizes the form; otherwise, the size of the rows and columns will change with the form, causing new printing to be inconsistent with earlier printing (unless you make the form nonsizable)
- You must call this routine each time you want to print to a form, not just once for each form at the start of your program

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNum("VnplChar.Width",4000)
'VnSetNum("VnplChar.Height",4500)
'VnSetRowsCols("VnplChar",20,60)
```

See Also

'VnInputScreen
'VnPrintAt
'VnPrintBox
'VnPrintCtrl
'VnPrintTo

7.4.58 'VnSetSysColor

Syntax

FUNCTION 'VnSetSysColor (
  /POINTER *_Index* **)**

Parameters

_Index
  Index of the system color

Description

This function gets the color value for one of the system colors. The system colors are as follows:

`_VnActiveBorder`	`_VnInactiveBorder`
`_VnActiveTitleBar`	`_VnInactiveTitleBar`
`_VnAppWorkspace`	`_VnMenuBar`
`_VnButtonFace`	`_VnMenuText`
`_VnButtonShadow`	`_VnScrollBars`
`_VnButtonText`	`_VnTitleBarText`
`_VnDesktop`	`_VnWindowBackground`
`_VnGrayText`	`_VnWindowFrame`
`_VnHighlight`	`_VnWindowText`
`_VnHighlightText`	

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetNum("MainForm.BackColor",
            'VnSetSysColor(_VnDesktop))
```

See Also

'VnGetColor
'VnSetRGBColor

7.4.59 'VnSetTran

Syntax

PROCEDURE 'VnSetTran (
/POINTER *_TranPairs$* **)**

Parameters

TranPairs$
A new translation table

Description

This procedure sets the current font translation table. The translation table consists of zero or more pairs of characters in which the first is the NPL character and the second is the VB character.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
'VnSetTran("AaBbCc")
```

See Also

'VnGetTran$

7.4.60 'VnSleep

Syntax

PROCEDURE 'VnSleep

Description

This procedure puts the NPL run time to sleep until `VnWakeup` is called. Generally, `'VnWakeup` is called in response to some event, such as the clicking of a *Close* button. This procedure actually does a `KEYIN` statement in order to stop the run time from executing. While in a `KEYIN`, NPL procedures can still be called from an external library (like **`VNPL16.DLL`**), allowing NPL to process events from your VB program. Eventually, one of these procedures must call the `'VnWakeup` procedure.

Warning The NPL window should *not be visible* when your program invokes `'VnSleep`. If it is, the user will be able to type a character or click the mouse within the NPL window, both of which will cause the `KEYIN` statement to finish and `'VnSleep` to resume.

Return Values

Value	Meaning
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
;Display the form modelessly
'VnMethod("HelloForm.Show"," ")
;
;Wait until 'VnWakeup is called
'VnSleep
;
;Unload the form
'VnCmd("HelloForm","Unload"," ")
;
;Close the link to the VB app
'VnClose
```

See Also

'VnWakeup

7.4.61 'VnWakeup

Syntax

PROCEDURE 'VnWakeup

Description

This procedure wakes up the NPL run time after an earlier call to `'VnWakeup`. Generally, `'VnWakeup` is called in response to some event, such as the pressing of a *Close* button. To the run time program, this procedure simulates the pressing of a key Because the `'VnSleep` procedure does a `KEYIN`, this causes it to return from the `KEYIN` and then return to the NPL program that called it, effectively waking up the NPL program.

Return Values

Value	**Meaning**
`VnError`	Returns 0 if successful, otherwise, returns a >0 value

Example

```
PROCEDURE 'CloseButtonClick/PUBLIC
;
'VnWakeup
;
END PROCEDURE 'CloseButtonClick
```

See Also

'VnSleep

C H A P T E R 8

VB Reference

This chapter contains:

- Descriptions of each of the VB constants
- A list of the VB subroutines categorized by the type of operation they perform
- Definitions and detailed descriptions of each of the VB subroutines in alphabetical order

8.1 Constants

This section gives an overview of the constants defined in the **VnplUtil** module.

8.1.1 Version Number

The Visual NPL version number constant is:

```
VN_VERSION NUMBER
```

8.1.2 Error Codes

The Visual Basic error codes constants are as follows:

`VN_ERR_NO_NPLWND`	No NPL Window was found.
`VN_ERR_NO_APPNAME`	No application name was specified.
`VN_ERR_NO_MEMORY`	The internal message buffer couldn't be allocated.
`VN_ERR_DATA_TOO_LARGE`	The data is too large for the internal message buffer.
`VN_ERR_BAD_CONNECT`	Invalid NPL connection.
`VN_ERR_BAD_VERSION`	Version number mismatch between `VNPLUTIL.BAS` and `VNPL16.OCX`.
`VN_ERR_CANT_SET_APPNAME`	Can't set the application name.
`VN_ERR_CALLBACK_BAD_VAR`	Can't convert variant into NPL parameter.
`VN_ERR_CALLBACK_NOT_FOUND`	Can't find NPL procedure.
`VN_ERR_CALLBACK_BAD_NUM_PARMS`	Wrong number of parameters for NPL procedure.
`VN_ERR_CALLBACK_BAD_RETURN_TYPE`	Can't call NPL functions, just procedures.
`VN_ERR_CALLBACK_BAD_PARM`	Invalid parameter for NPL procedure.
`VN_ERR_CALLBACK_CANT_CALL`	Can't call NPL procedure.

8.2 Subroutines

The following table lists the VB subroutines grouped according to the type of operation they perform:

Operation	Subroutine
Application Starting Point	`Main`
Calling NPL Procedures	`VnCallProc`
Form Centering	`VnCenter`
Change-List Creation	`VnChg`
	`VnChk`
	`VnClose`
	`VnHot`
	`VnKey`
	`VnKeyPress`
	`VnMenuClk`
Event-Driven Support	`VnWakeup`
Error Handling	`VnErrMsg`
	`VnKill`
Support Functions Called Indirectly From NPL	`VnCtrlType`
	`VnDevDef`
	`VnSetCtrl`
	`VnSetObj`

Table 8.8 VB Subroutines by Type

8.2.1 Main

Syntax

Sub Main()

Description

This is the main procedure for the Visual Basic program. It does the initialization of the link between VB and NPL. When creating a new VB program you must specify this as the main procedure by selecting the *Options* item under the *Tools* menu, and under the *Project* tab, setting the **Startup Form** to ***Sub Main***.

This procedure is called automatically by Visual Basic (when set as described in the preceding section) when your program starts running. It should not be called directly from your NPL program.

Return Values

None

8.2.2 VnCallProc

Syntax

Function VnCallProc %(
ProcName$,
Parm1, ... ParmN **)**

Parameters

ProcName$
Name of the NPL procedure to call

Parm1
First parameter passed to this procedure

ParmN
Last parameter passed to this procedure

Description

This function is used to call an NPL procedure from VB. You must pass the name of the NPL procedure to be called, followed by any parameters required by the procedure. The NPL procedure must be declared as `PUBLIC`, the number of parameters must be correct, and the type of each parameter must match the type of data being passed.

Although **VnCallProc** is declared as a function, you can also call it as a procedure (as you can do with any VB function). In this case, VB discards the value that was returned. This is useful when you know the NPL procedure exists and you know what types of parameters it takes, which is typical when you are writing a computer program. So, in most cases you should treat **VnCallProc** as a procedure and call it accordingly.

There are really only two times when you might want to call **VnCallProc** as a function:

- When you don't know if the procedure exists, which should be a very rare occurence,
- When the NPL procedure might not be callable because your NPL program is doing some processing and not waiting for a callback by means of NPL procedures `'VnCmd` or `'VnSleep`.

Regardless of how you call it, for all other types of errors, **VnCallProc** will show a message box describing the error. This is because the other errors have to do with parameter mismatches, which is a development-time problem that should be solved long before a customer ever sees an application. In other words, all of these problems should be solved by the developer and the user should never see the message boxes.

Return Values

Value	**Meaning**
`0`	If successful
`Nonzero value`	If an error occurs

Example

```
DIM Name$
Rem VB

VnCallProc "Prompt","Customer Name","Enter name:"  ,Name

....


PROCEDURE  'Prompt(/POINTER _Title$,
                  /POINTER _Prompt$,
                  /POINTER Name$)/PUBLIC
  ;NPL
  ;
  Name$='VnInpBox$(_Title$ ,_Prompt$,0,Name$)
  ;
END PROCEDURE 'Prompt
```

See Also **VnWakeup**

8.2.3 VnCenter

Syntax

Sub VnCenter (
Frm As Form **)**

Parameters

Frm
A form to be centered on the screen

Description

This procedure centers a form on the screen. It does this by setting the Top and Left properties of the form based on the current sizes of the screen and the form. This procedure does exactly the same thing as the NPL `'VnCenter` procedure. It is provided for the sake of convenience.

Return Values

None

Examples

```
Private Sub Form_ Load()
    Rem VB
    VnCenter Me
End Sub
```

See Also

NPL procedure **'VnCenter**

8.2.4 VnChg

Syntax **Sub VnChg()**

Description This procedure sets an internal flag indicating that the current control has been changed in some way. It is used to indicate that the control's new value should be added to the change-list when `VnChk` is next called.

Return Values None

Example

```
Private Sub Address_ Change()
    Rem VB
    VnChg
End Sub

Private Sub OKButton_ Click()
    Rem VB
    VnChg
    VnHot
End Sub
```

See Also **VnChk**
VnHot

8.2.5 VnChk

Syntax

Sub VnChk()

Description

This procedure checks to see if the current control has been changed in some way. If so, the control's new value is added to the change-list and the control's Tag property is checked to see if it is "Hot". If it is, the change-list is sent back to NPL.

Return Values

None

Example

```
Private Sub Address_ LostFocus()
    Rem VB
    VnChk
End Sub
```

See Also

VnChg
VnHot

8.2.6 VnClose

Syntax

Sub VnClose(
    UnloadMode **%)**

Parameters

UnloadMode
    UnloadMode parameter from a **QueryUnload** event

Description

This procedure is meant to be called from a **QueryUnload** event procedure. It determines if the form is being closed from code (that is, by an **Unload** command) or by some other means. If the form is being closed from code, then this call does nothing. Otherwise, it sends the change-list back to NPL so that your program can react to the form being closed. It adds an entry to the change-list with the `Flag$` field set to "X" and the `Control$` field set to "Form Close".

Return Values

None

Example

```
Private Sub Form_ QueryUnload(Cancel%, UnloadMode%)
    Rem VB
    VnClose UnloadMode
End Sub
```

See Also

NPL procedure '**VnWakeup**

8.2.7 VnCtrlType

Syntax

Function VnCtrlType $(
Ctrl As Control **)**

Parameters

Ctrl
Any VB control

Description

This is a developer-modifiable routine that can be found in the ***VnplDev*** module in your VB project. As such, you are responsible for its contents.

This function is used by the NPL `'VnGetFormCtrlList` procedure to get a 3 character type name for a control. For the standard controls (that is, those that are part of all VB projects), the control name prefixes found on pages 40-41 of the *Microsoft Visual Basic Programming System form Windows, Version 4.0 Programmer's Guide* are used. For the standard controls that are optional, and for the 3-D versions of the standard controls, the lines defining the controls' type names are commented out. Remove the comment delimiters for the controls that you are using in your project.

For all other controls, you need to add VB code to return the type name. The routine consists mainly of a large set of nested `If` (that is, `ElseIf`) statements, so add a new `ElseIf` at the end as follows:

```
ElseIf TypeOf Ctrl Is ControlName Then
    VnCtrlType = "XXX"
```

Replace `ControlName` with the name of the control and replace "XXX" with any three-character type name that doesn't conflict with any other type name in the procedure.

This procedure is called automatically by Visual Basic when your NPL program calls the NPL `'VnGetFormCtrlList` procedure. It should not be called directly.

Return Values

This function returns a three-character type name for the control.

Example

See above.

See Also

NPL procedure **'VnGetFormCtrlList**

8.2.8 VnDevDef

Syntax

Function VnDevDef %(
Cmd$,
Obj As Object,
ObjValue$ **)**

Parameters

Cmd
Developer-defined command

Obj
Object on which the command operates

ObjValue
Command-specific parameter values

Description

This is a developer modifiable routine that can be found in the ***VnplDev*** module in your VB project. As such, you are responsible for its contents.
This procedure is used to create developer-defined commands for use with the NPL `'VnCmd` procedure. Whenever `'VnCmd` doesn't recognize the command that it's been given, it calls this procedure with the parameters that it was passed. In other words, you can create any command you want by adding code to `VnDevDef`.

For each command you want to create, add a `Case` to the `Select` statement in the `VnDevDef` function. By default, the `Dev Def` command shows its parameter in a message box and returns the result in the parameter:

```
' dummy "dev def" command
Rem VB
Case "dev def"
    ObjValue = MsgBox(ObjValue)
```

This command could be called from NPL program as follows:

```
 'VnCmd(" ","Dev Def","Hello Vinny!")
```

Because this is a sample command, feel free to delete it or use it as the starting point of your first command. Notice that the command name is not case sensitive and that the object parameter need not be passed. In general, you can define the parameters in whatever manner you find appropriate.

This procedure is called automatically by Visual Basic when your NPL program calls the NPL `'VnCmd` procedure with a nonstandard command. It should not be called directly.

Return Values

Value	Meaning
`0`	If successful

`Nonzero value`	If an error occurs

Example See above

See Also NPL procedure**'VnCmd**

8.2.9 VnErrMsg

Syntax

Function VnErrMsg$(
ErrCode% **)**

Parameters

ErrCode
Visual NPL error code

Description

This function returns the error message corresponding to a specific Visual NPL error code. It is used by the VB **Main** procedure to report any errors while trying to initialize the connection with NPL.

Return Values

The error message corresponding to the error code.

Example

```
Result = VnplLink.VnCon.Init(App.EXEName)
Rem VB
If Result <> 0 Then
   Beep
   MsgBox VnErrMsg(Result), vbCritical, "Error"
   End
End If
```

See Also

Main

8.2.10 VnHot

Syntax

Sub VnHot()

Description

This procedure can be called in change-list programming instead of calling `VnChk` and setting individual Tag properties to "Hot". Normally all hot controls have their **Tag** property set to "Hot". Every time `VnChk` is called, the Tag property is checked; if it's "Hot" then control is passed back to NPL. `VnHot` performs the same operation but eliminates the need to set the Tag property. You can call `VnHot` for each hot control instead of calling `VnChk`.

Return Values

None

Example

```
Private Sub OKButton_ Click()
    Rem VB
    VnChg
    VnHot
End Sub
```

See Also

VnChg
VnChk

8.2.11 VnKey

Syntax

Sub VnKey(
KeyCode%,
Shift% **)**

Parameters

KeyCode
KeyCode parameter from a **KeyUp** or **KeyDown** event

Shift
Shift parameter from a **KeyUp** or **KeyDown** event

Description

This procedure is meant to be called from a **KeyUp** or **KeyDown** event procedure to send the keystroke to NPL as a single-item change-list. The entry in the change-list will have the `Flag$` field set to "K" and the `ChgValue$` field set to the string versions of the `KeyCode` and `Shift` parameters. The `Shift` parameter is always a single-digit number that will appear as the last character of the field.

Return Values

None

Example

```
Private Sub Form_ KeyUp(KeyCode%, Shift%)
    Rem VB
    VnKey KeyCode, Shift
End Sub
```

See Also

VnKeyPress

8.2.12 VnKeyPress

Syntax

Sub VnKeyPress (
KeyAscii% **)**

Parameters

KeyAscii
KeyAscii parameter from **KeyPress** event

Description

This procedure is meant to be called from a **KeyPress** event procedure to send the keystroke to NPL as a single item change-list. The entry in the change-list will have the `Flag$` field set to "A" and the `ChgValue$` field set to the string version of the `KeyAscii` parameter.

Return Values

None

Example

```
Private Sub Form_ KeyPress(KeyAscii%)
    Rem VB
    VnKeyPress KeyAscii
End Sub
```

See Also

VnKey

8.2.13 VnKill

Syntax **Sub VnKill**

Description This procedureis meant to be called from the Debug window when your VB program stops and can’t continue. It will make the NPL window visible, re-enable it, and then close the connection between NPL and VB.

Warning Do not use this procedure in your program. It is intended as a developer’s tool to be used only when things go wrong.

Return Values None

Example `VnKill`

See Also NPL procedure'**VnClose**

8.2.14 VnMenuClk

Syntax

Sub VnMenuClk (
MenuName$ **)**

Parameters

MenuName
The name of the menu command being selected

Description

This procedure is meant to be called from a menu command event procedure to send the menu command and the current change-list back to NPL. The menu command's entry in the change-list will have the `Flag$` field set to "C" and the `Control$` field set to the `MenuName` parameter. The `MenuName` parameter can be any name you want; it doesn't have to be the name of the menu control.

Return Values

None

Example

```
Private Sub FileMenu_ Click(Index%)
    Rem VB
    Select Case Index
        Case 1
            VnMenuClk "FileNew"
        Case 2
            VnMenuClk "FileOpen"
        Case 3
            VnMenuClk "FileSave"
        Case 4
            VnMenuClk "FileSaveAs"
        Case 5
            VnMenuClk "FilePrint"
        Case 6
            VnMenuClk "FileExit"
    End Select
End Sub
```

See Also

VnChg
VnChk
VnHot

8.2.15 VnSetCtrl

Syntax

Function VnSetCtrl %(
Obj As Object,
CtrlName$,
Index%,
Ctrl As Control **)**

Parameters

Obj
The form on which the control array exists

CtrlName
The name of the control array

Index
The index into the control array

Ctrl
The control from the control array

Description

This is a developer-modifiable routine that can be found in the ***VnplDev*** module in your VB project. As such, you are responsible for its contents.

This function is used by the NPL '`VnCreateCtrls` procedure to do the actual control creation. For each control array on each form on which you create controls, add a `Case` to the `Select` statement in the `VnSetObj` function. The text of the `Case` should be the name of the control array and the single line of code for the `Case` should set the `Ctrl` parameter to element `Index` of the control array:

```
Case "TextBox"
    Set Ctrl = Obj.TextBox(Index)
```

This procedure is called automatically by Visual Basic whenever your NPL program calls the `VnCreateCtrls` procedure. It should not be called directly.

Return Values

Value	Meaning
`0`	If successful
`Nonzero value`	If an error occurs

Example

See above.

See Also

NPL procedure '**VnCreateCtrls**
VnSetObj

8.2.16 VnSetObj

Syntax

Function VnSetObj %(
ObjName$,
Index%,
Obj As Object **)**

Parameters

ObjName
The name of the object

Index
The index of the object if it's a member of an array

Obj
The object (or array of objects)

Description

This is a developer-modifiable routine that can be found in the ***VnplDev*** module in your VB project. As such, you are responsible for its contents.
This procedure is used to register the top-level objects (mostly forms) of your VB program so that your NPL program can access them. For each object that you want to use, add a `Case` to the `Select` statement in the `VnSetObj` function. The text of the `Case` should be the name of the object and the single line of code for the `Case` should set the `Obj` parameter to the object itself as follows:

```
Case "MainForm"
    Set Obj = MainForm
```

This object can now be used in the NPL program, for example:

```
PRINT 'VnGetAlf$("MainForm.Caption")
```

This procedure is called automatically by Visual Basic whenever your NPL program passes an object to one of the NPL procedures. It should not be called directly.

Return Values

Value	Meaning
`0`	If successful
`Nonzero value`	If an error occurs

Example

See above.

See Also

VnSetCtrl

8.2.17 VnWakeup

Syntax

Sub VnWakeup (
UnloadMode% **)**

Parameters

UnloadMode
UnloadMode parameter from a **QueryUnload** event

Description

This procedure is meant to be called from a **QueryUnload** event procedure. It determines if the form is being closed from code (that is, by an **Unload** command) or by some other means. If the form is being closed from code, this call does nothing. Otherwise, it calls the NPL's `VnWakeup` procedure to wake up the NPL program so that it can react to the form being closed.

Return Values

None

Example

```
Private Sub Form_ QueryUnload(Cancel%, UnloadMode%)
    Rem VB
    VnWakeup UnloadMode
End Sub
```

See Also

NPL procedure '**VnWakeup**
VnCallProc
VnClose

Index