The Visual Basic - P-NET Connection

Christopher Jenkins - Proces-Data (UK) Ltd.

The Paper is divided into three parts. Firstly, a brief overview of Visual Basic is given, including illustrative examples of standard controls and their use. Secondly, the use of internal variables to control a graphical object, will be described. Finally, using the principles previously described, the methodology for declaring external object variables and the communication of data between a P-NET module and a Visual Basic Application will be demonstrated.

Visual Basic - A brief overview

Visual Basic has become a very popular and important Application Development Language in the last couple of years. This is because:
:
1) It applies specifically to one of the most well known graphical operating systems - namely MicroSoft Windows, for the purposes of developing Windows application programs.

2) It is designed with the Application Writer in mind, rather then the high flying programming elite.

Visual Basic can be defined as an Object Orientated, Procedural Language. This means that graphical images, documents, programs, and pieces of hardware can be treated as individual entities, and as such, can be operated upon by a program using sub procedures or functions.

Visual Basic is a very rich language, in that it not only supports all the usual features of an application language, but has many additional features not normally found in the traditional *standard* languages.

It supports simple and complex Data Types, Arrays and Files, and if there is any familiarity with Standard Basic, it is easy to learn. Be warned however, that it still retains the ubiquitous GOTO Statement! However, it is far more powerful than the standard Basics, in that it has been designed to produce programs for the Graphical environment.

One of the major advances within the Windows environment has been the development of Object Linking and Embedding technology - OLE, where objects such as a database, spreadsheet, text or even sound, from one application, can be controlled or embedded in another application.

All major MicroSoft Applications support OLE, e.g. WORD, Excel, Access, and competitive products from other companies such as Novell, with WordPerfect, Quatro Pro (Spreadsheet) & Paradox (Database), have followed suit. Programs written in Visual Basic can act as both OLE Client and Server applications.

Furthermore, the macro language of all the major MicroSoft applications, uses a subset of Visual Basic, called Visual Basic for Applications. This means that familiarity with Visual Basic makes it easier to write Macros for standard Windows packages, as well as providing the means to write stand alone applications.

Visual Basic has access to the Windows API (Application Interface), which are a collection of Function libraries called Dynamic Link Libraries.. This means that a Visual Basic program can make calls on nearly all these pre-written functions, the vast majority of which are written in "C", without the need to re-invent the wheel. For example, there is no direct means in Visual Basic to flash the Title of a Window, but there is a function in the API. Other libraries can also be used, and we have already heard of the HUGO API.

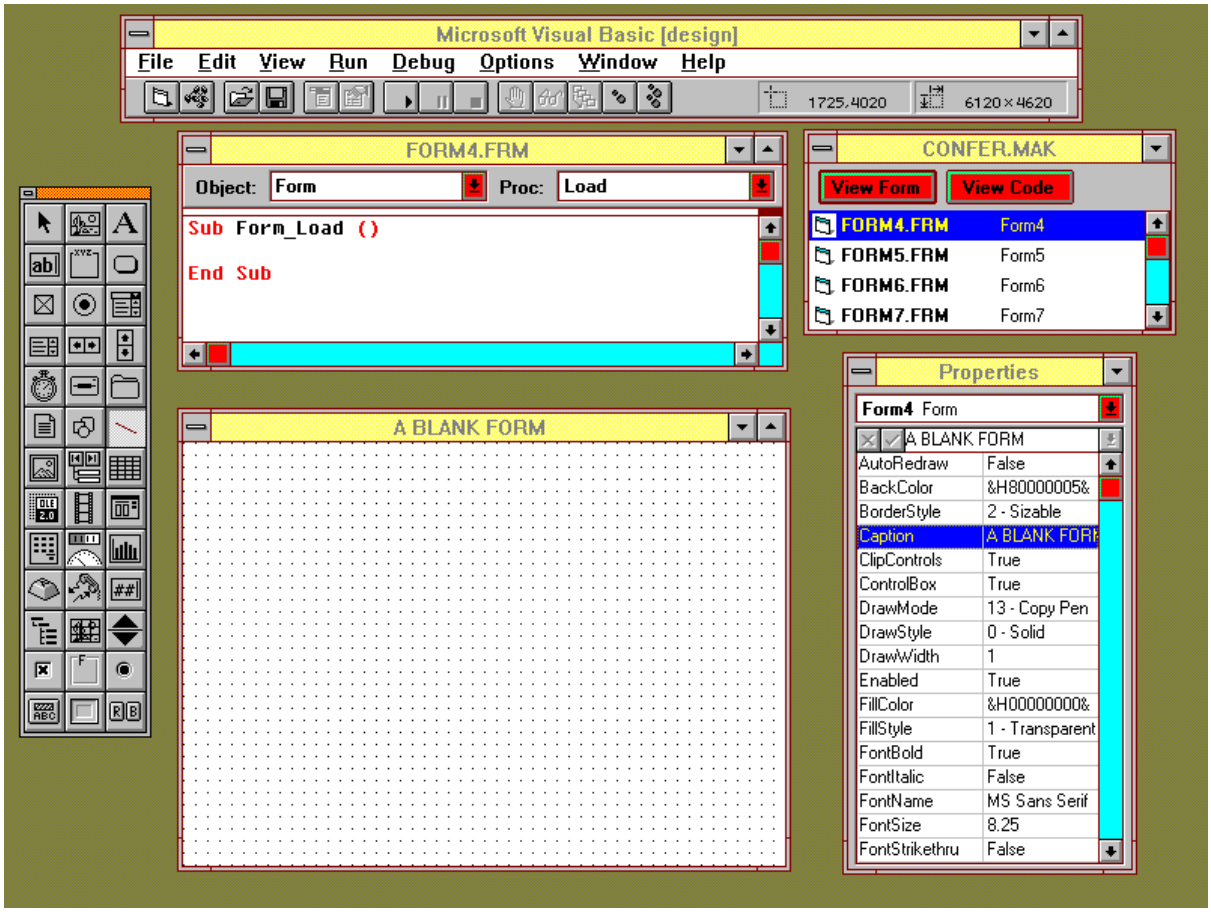
The fact that a Windows application is being developed, means that all other Windows utilities are available to it, e.g. Print Manager, Scalable Fonts etc., thus eliminating additional development work.

Visual Basic has a number of built in graphical objects to form the basis of an application, including a helpful menu design facility, and there are a wide range of User Objects available from third party suppliers.

Once the application has been designed, Visual basic offers services to provide an integrated HELP facility, and allows an installable, executable package to be generated, for customer distribution.

The Design Screen

Unlike traditional programming languages, Visual Basic provides a graphical environment for program design, in the form of a number of Design Windows. Under normal circumstances, these would be overlapping, but as with other Windows applications, can be re-arranged and re-sized to suit the users requirements.



First we have the Controls Window or Toolbox, which allows the designer to pick one of the available standard or special Controls, and position it on a FORM. Normally, one would double click a Control, which will then appear in the centre of the Form. The designer can then drag and resize the control.

The FORM Window is the Container for all controls associated with that form. There can be a number of Forms within a PROJECT or application.

All OBJECTS within the graphical environment have a number of associated properties or attributes which can be given a value. The PROPERTIES Window provides an overview of the PROPERTIES of the selected Control Object. Notice also, that some properties can be determined automatically, during the drawing stage, or can be set at design time, by typing into the appropriate part of the properties list, or can be controlled by program code at run time.

The PROJECT Window has three functions. Firstly, it acts as a definition of how the project

is structured, in terms of the numbers of forms and external control objects. Secondly, it acts as a means of selecting a particular form for display, and thirdly, as a means of viewing and writing code.

The CODE WINDOW, when it has the FOCUS, provides the programmer with the means to write and edit specific sections of code.

The MENU BAR provides all the facilities for saving, printing and debugging the program.

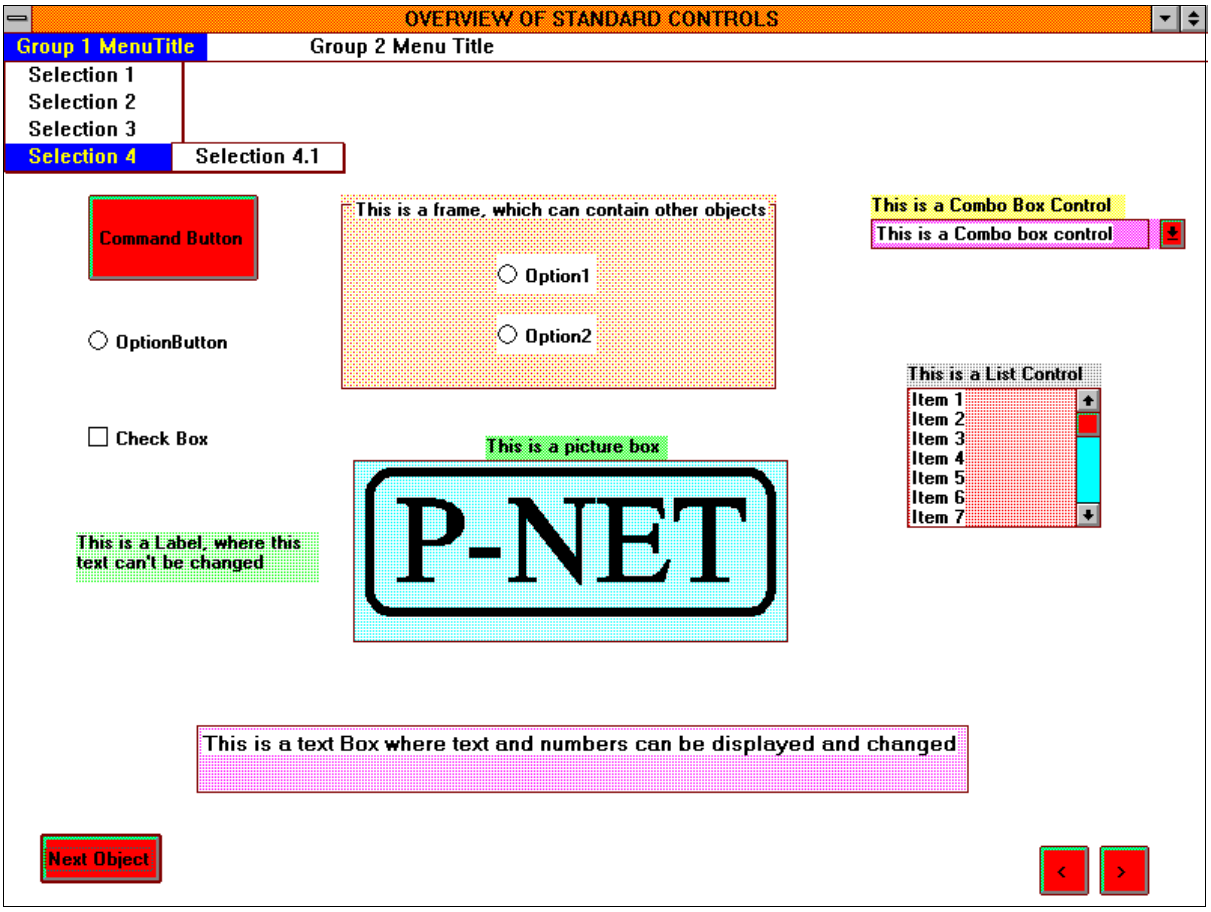
Standard Objects

The Form

A displayed window is based on a FORM. A Form normally includes a TITLE BAR, CONTROL BOX and Maximise and Minimise controls. The Form is a container for all other Controls.

There are many different types of Standard controls. There are many more customised controls available on the market, which you can include in your application.

Here are some standard ones.



The Command Button

The Command Control is probably the most common control in many applications. By instigating an EVENT on this object, such as clicking on it with a mouse button, a piece of program code in the form of a SUB-PROCEDURE is invoked.

The Frame

The frame is another container which can have a caption and allows other controls to be moved or acted upon independently of other controls on the form.

Option Box

This familiar control allows the user to instigate one of a number of choices. Additional option arrays have to be contained in their own frame.

Check Box

Similar to the Option Control, but allows multiple boxes to selected at the same time.

Text Box

This allows alpha-numeric text to be input by the user, or acts as a display area for variable values.

Label Box

Similar to the Text Box, but doesn't allow user input.

Combo Control

Provides the means to select form a drop down list of choices, and display the choice.

List Box

Provides a permanent list of selections, one of which is highlighted

Picture Box

Provides a container for a graphical image.

Menu

Another familiar facility to be found in most applications, provides a number of choices, the selection of one will activate an event procedure.

Common Dialogue

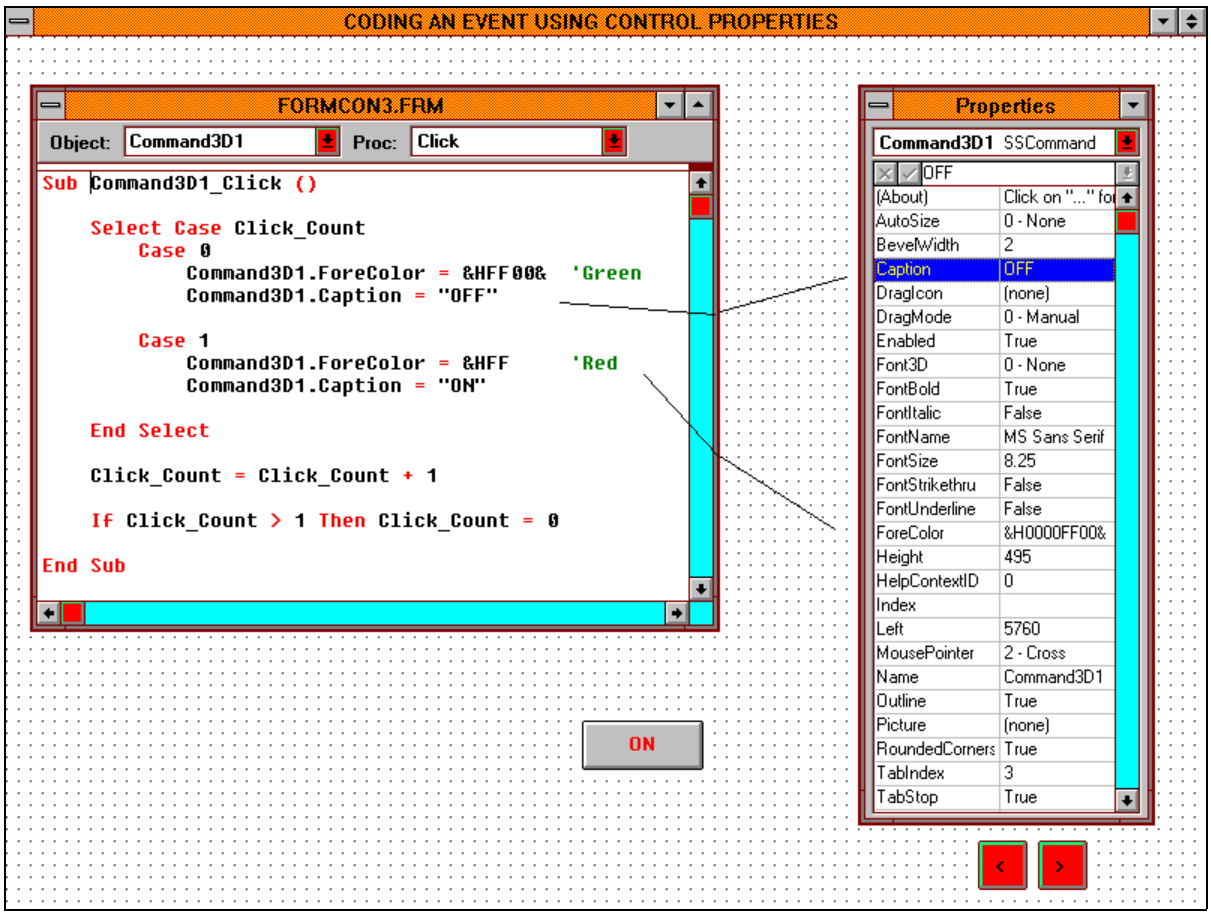
One of the compound controls available, familiar to all Windows users. The properties associated with this control, allow it to be configured for such services as Saving or Opening a file, printing or selecting a font etc.

Message Box

Another familiar compound control, usually configured for some sort of error handling, and provides a programmed message and a selection of control buttons.

Coding an Event

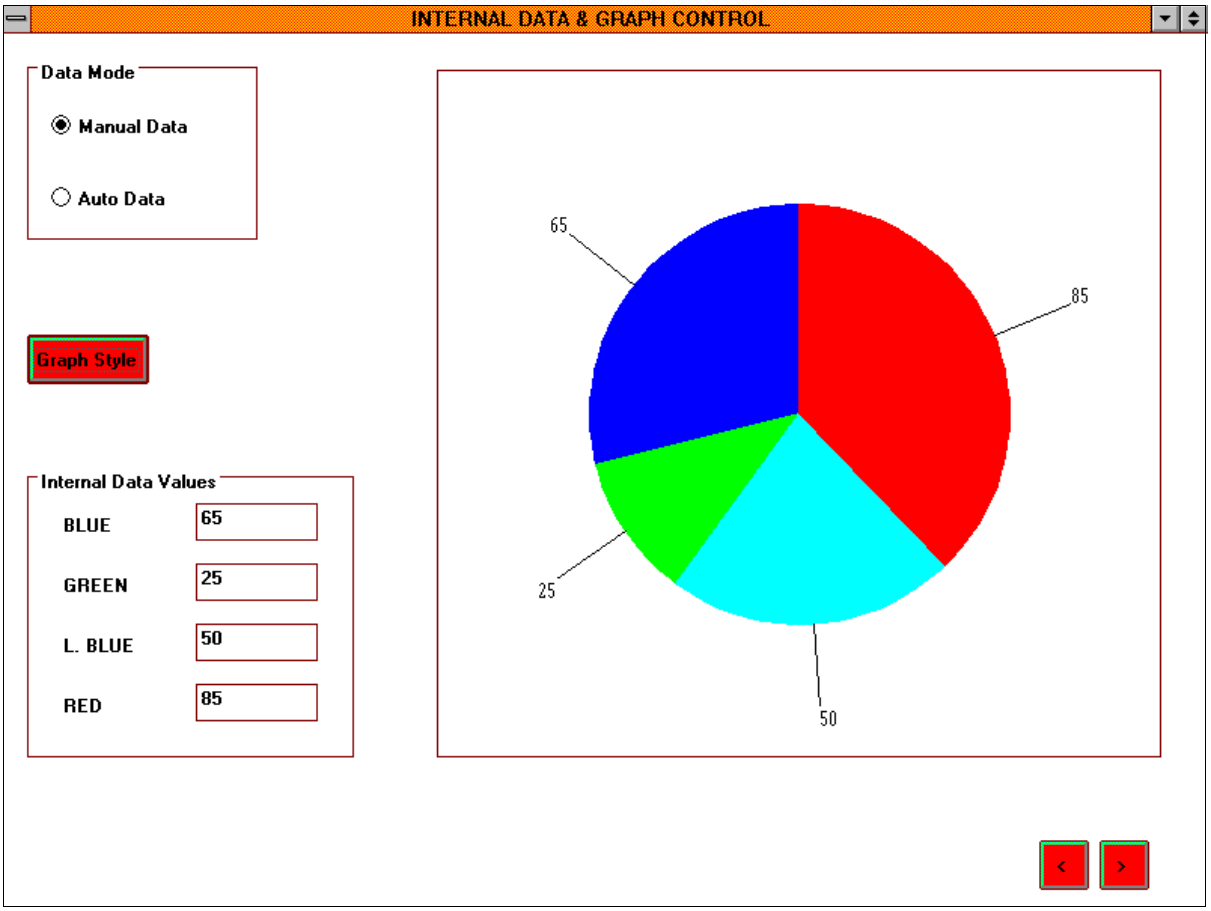
Here is a simple example of a Command Button, which when clicked, changes its caption. Notice that at *Design Time*, the initial properties of the Command Button can be defined. At *Run Time*, the *Caption* property is changed by means of code, stimulated when the object is clicked. Other EVENTS associated with this object could have been chosen, such as double click or drag.



Assigning Variables to Objects

So how do we associate variables with our application.

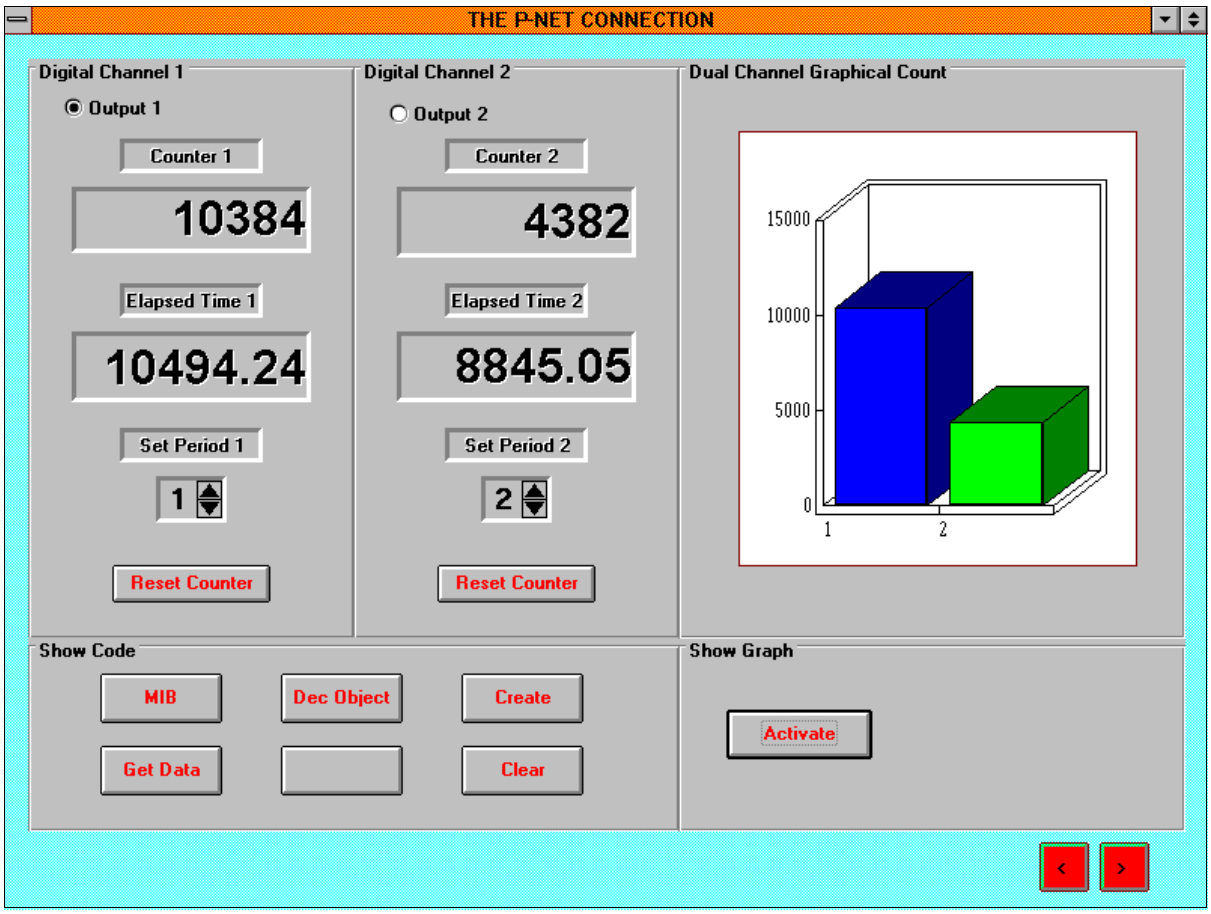
Here is a form with some text boxes which are displaying the value of four predefined internal variables. Also included are a couple of command buttons, an option group in a frame, and a User Control called Graph.VBX.



By manually inputing values for these variables, I can make the graph control *GraphData* property values change. I can also change the style of the graph by changing the *GraphType* property value, by activating a click sub-procedure with a Command control button. This isn't particularly exciting, so rather than inputing numbers manually, I have written some code to increment the variables on the basis of a Timer Control. This shows that the screen can be controlled by means of defined internal variables, so how do we get data from the outside world to do the same thing?

Using VIGO to Declare External Objects

Here is a form containing a pseudo application. What is required is to display a number of external variables via P-NET from two channels of a digital module PD 3120.

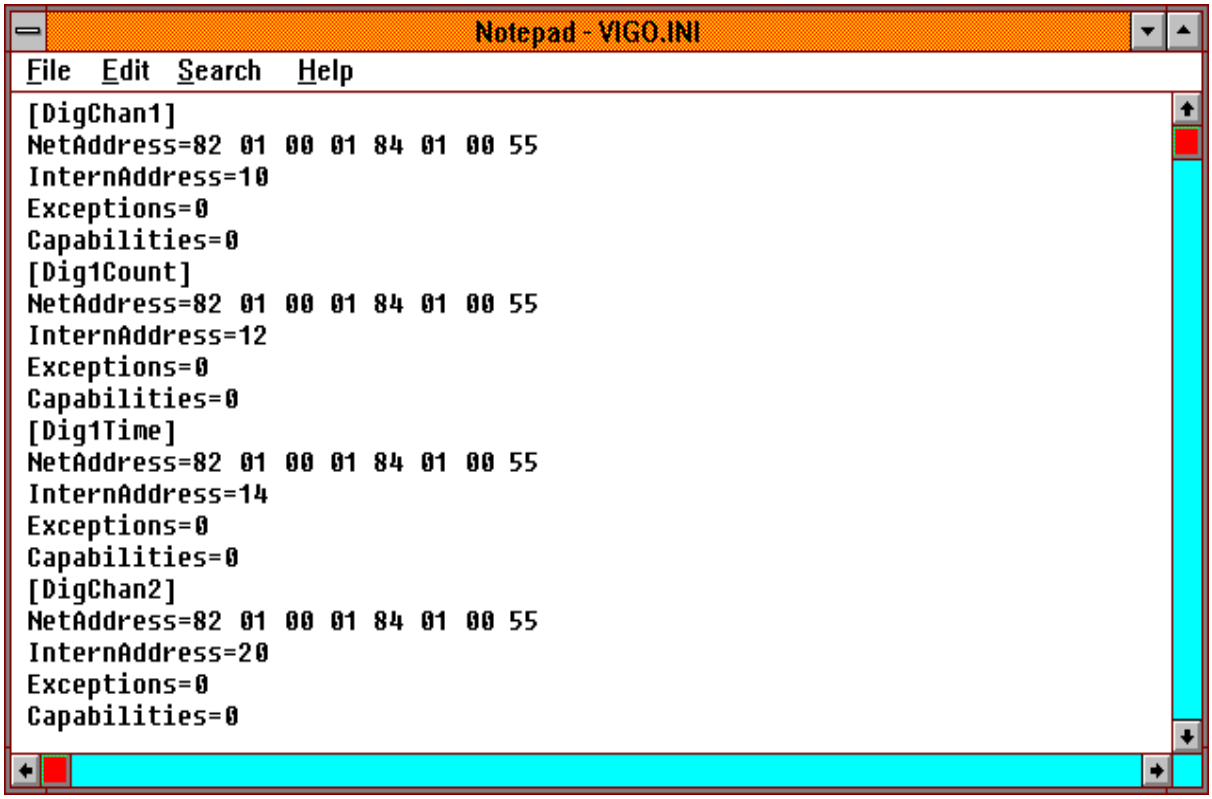


From each channel, three data types have been selected. A *Boolean*, which is the Flag Register, indicating the state of the output, an *Integer*, which is the value of the output counter, and a *Real*, which is the state of the OperatingTime register.

There is a program running in the Calculator Channel of the module, which merely cycles these two outputs with different time periods.

Notice an Option Button is being used to represent the actual state of the output, together with two Text Boxes. The *Font Size* Property of the text boxes have been changed from the default setting, at design time, to allow the numeric data to be displayed larger than normal.

As VIGO (VIrtual Global Object) is the means by which P-NET variables are to be connected to the Visual Basic application, it must be ensured that VIGO is aware of these variables, and how to get hold of them. A definition of these object variables must therefore be made within the Manager Information Base (MIB) in order that VIGO and HUGO can achieve real-time communication.



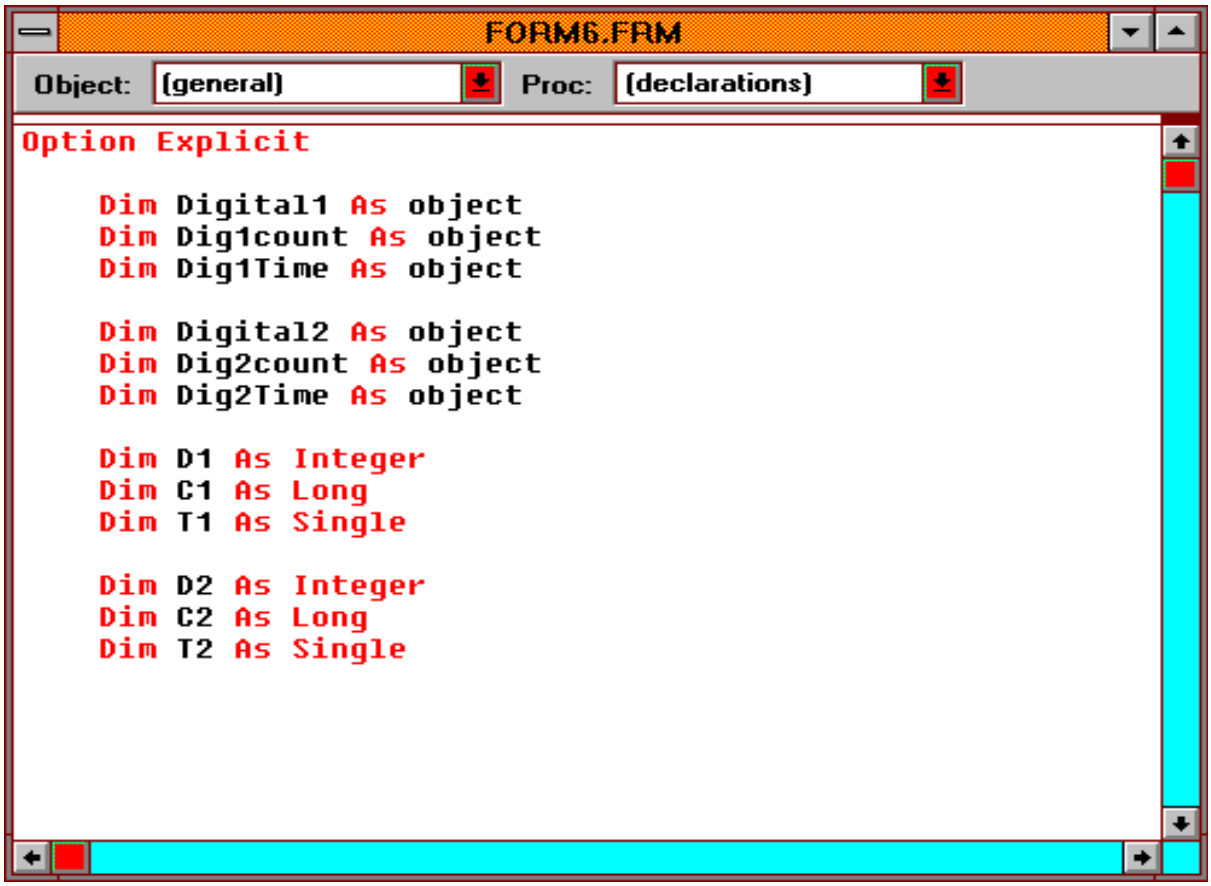
The *PhysID* property of each P-NET *Object* is included in this text file.

Looking at this list, we see that names have already been defined, which also include the source address.

We have already heard that this address may involve routes via other bridges or gateways, and may involve other network protocols and even other fieldbuses, although we don't want to mention any of those at *this* conference!

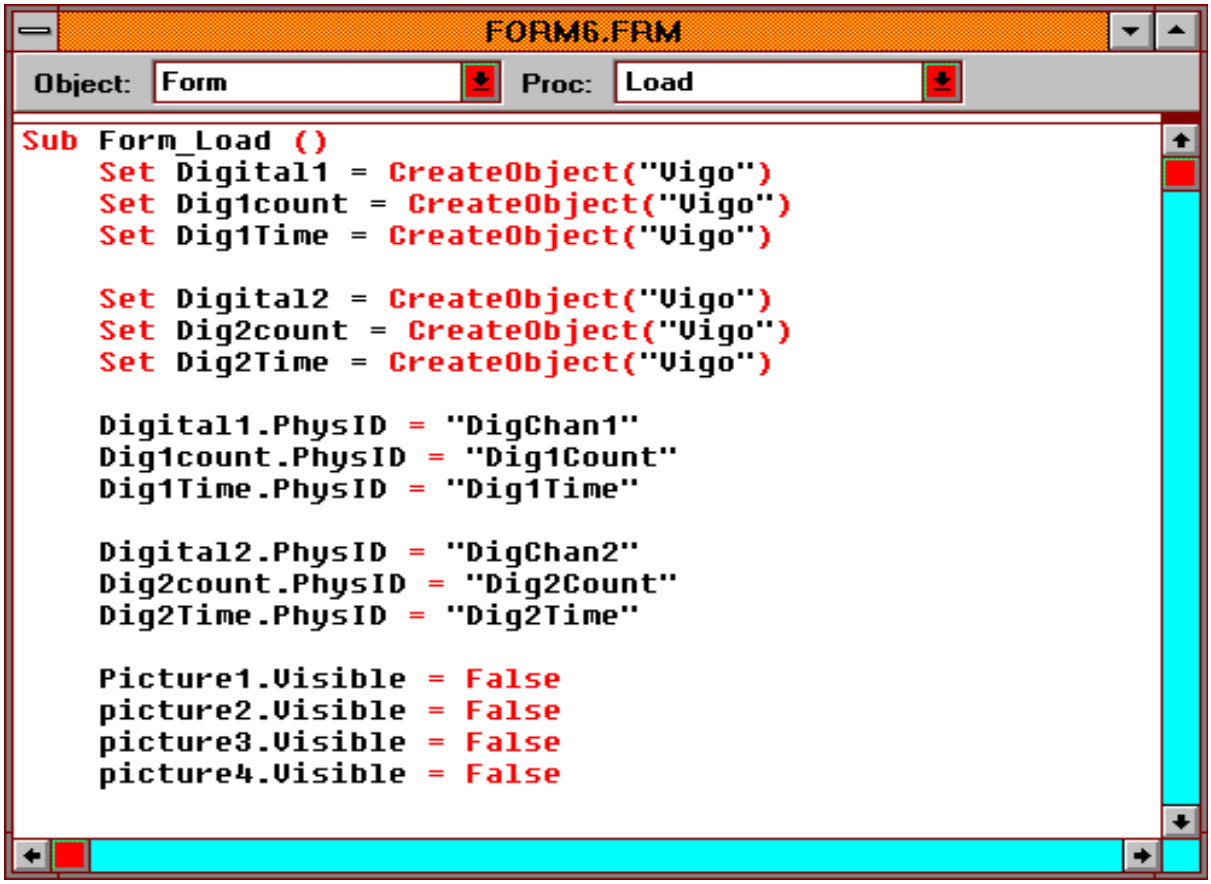
The P-NET Connection.

Firstly, for this form, I have defined the required P-Net variables as Objects, in the General Declaration section for the form. This means that the properties of these object variables can now be used in code.



Secondly, I have defined internal variables, which will be used to display data on the screen, in the text boxes.

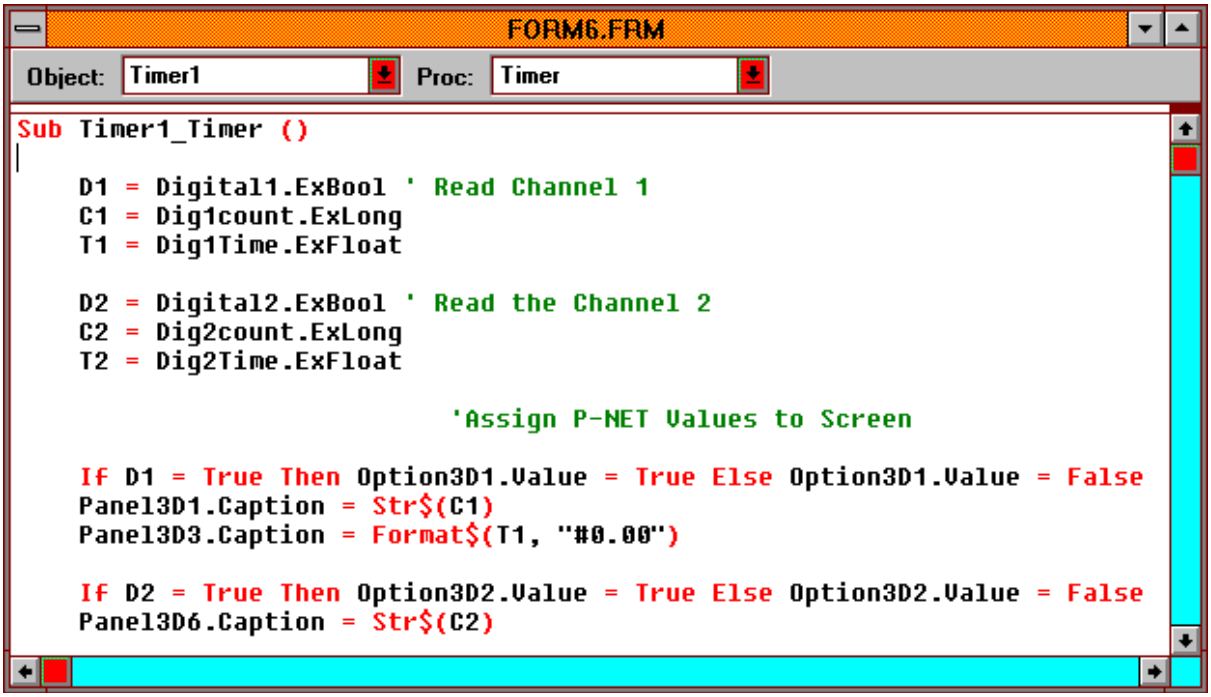
When the form loads, that is, when the form is selected for display, the sub-procedure *Form_Load* is invoked.



I am using the OLE methodology for setting up the connection between the Visual Basic program and P-NET via VIGO (and HUGO). Each new object variable must use the SET statement.

To ensure that the route to the object variable is known, I must define the Physical Identifier property of the object, as is know to the MIB

Now that the form is loaded, requests can be made via VIGO/HUGO to get, or modify the value of the defined variables.



Now that communication has been established, the variables can now be used like any other internal variable. For example, unlike the previous form, the Graph Control can now be utilised to graphically represent the value of the P-NET module counters.

Extending the Connection

Although outside the scope of this paper, the variables can now also be used, using OLE technology to transfer data to other Windows applications.