*em**Power*** ®

Programmer's Reference Manual

Part No. 09-0245

REVISION C

September 05, 2001

Document History

Rev A	**ECO xxxxx**	**6/28/00**	**B. H.**
Rev B	**ECO xxxxx**	**7/18/01**	**B. H.**
Rev C	**ECO 15390**	**9/05/01**	**B. H.**

Proprietary Rights Notice

This document and the information that it contains are the property of NH Research, Incorporated. The rights to duplicate or otherwise copy this document, the rights to disclose the document and its information to others, and the right to use the information therein may be acquired only by written permission signed by a duly authorized officer of NH Research, Incorporated.

Unauthorized duplication of software and documentation provided with the test workstations violates the copyright protection provided by law. However, backup copies of programs are permitted for archives and recovery from hardware failures.

The material in this publication is current as of the release date on the cover page and is subject to change without notice. Any questions concerning the product or any questions or comments concerning this manual should be directed to the NH Research Field Service Department during normal business hours Monday through Friday at (949) 474-3900 or FAX (949) 474-7062 or www.nhresearch.com.

*em**Power***®
is a trademark of NH Research, Incorporated.

TABLE OF CONTENTS

1. CLIENTS & SERVERS

Clients & *Servers* are cooperating applications – *Client* applications request *Services* from their *Server* applications, and the *Server* applications provide *Services* to their *Client* applications.

The *em**Power***® applications

*em**Power***® consists of four applications: ***Power****EDIT*, ***Power****EXEC*, ***Power****VIP*, *DistributeNHR*, and many *Components*. These *Components* are based on the *Microsoft Component Object Model* (COM), and are organized in a *Client*/*Server* relationship (see Figure 1). At the top is a *Client* that relies on one or more *Server* to help it do its job. Some *Servers* act as *Clients* to other *Servers*.

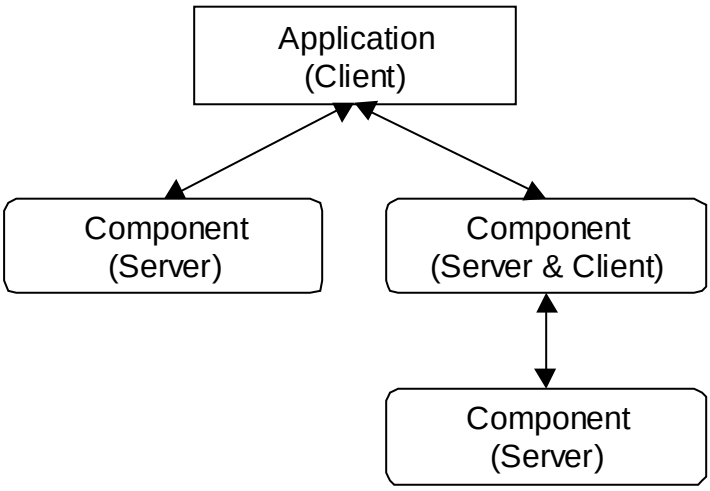

Figure 1 – The basic *Client*/*Server* relationship model

To help clarify the terminology used, *em**Power***® is the name of the total software system. It is also the name of the root application that is used to start all other applications used during the creation, running, debugging, and maintaining of test programs and related activities (see Figure 2).

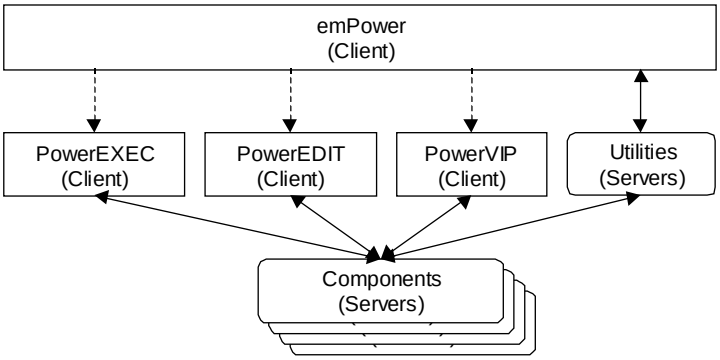

Figure 2 – The top level view of *emPower*** ®**

Component Object Model - COM provides a group of conventions and supporting libraries that allow interaction between different pieces of software in a consistent, object-oriented way (a *binary standard*, not a language). It is implementation language independent. COM defines an object concept known as an *interface*, which is a collection of methods that a client application can call. Each interface is based on the fundamental COM interface, ***IUnknown***. The methods of **IUnknown,** the base interface of every other COM interface, allow navigation to other interfaces exposed by the object. IUnknown defines three methods: ***QueryInterface***, ***AddRef***, and ***Release***. **QueryInterface** allows an interface user to ask the object for a pointer to another of its interfaces. **AddRef** and **Release** implement reference counting on the interface.

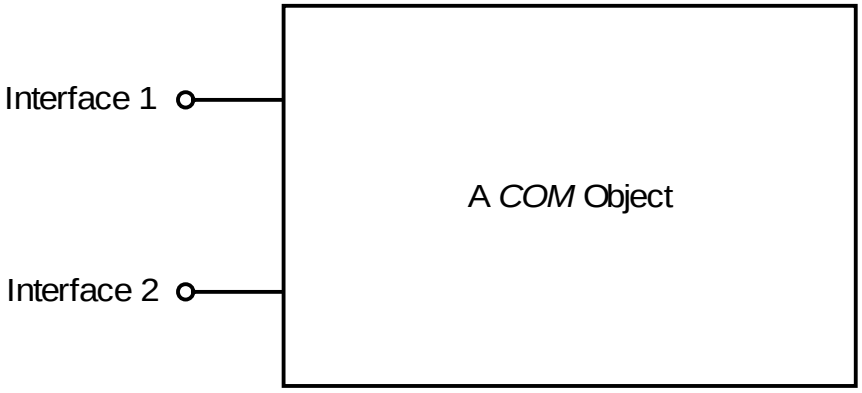

Figure 3 – The *Component Object Model* view

1.1 Key Points about COM

COM is a binary standard that was developed to facilitate component based application development. There are a few key concepts that must be understood to program with COM.

- Components have one or more interfaces.
- Interfaces are a set of properties and methods that can be used to interact with the component.
- Once published, an interface will NEVER change.
- Every interface has a unique identifier which will NEVER change.
- If there is a pointer to one interface in a component, then a pointer can point to any other interface in the component.

1.2 Common Interfaces and their Benefits

Many *em**Power*** components share one or more interfaces. For example, Loads and Sources expose the same DMM interface as does a measurement device. Because of this common syntax layer, you can learn the functions once and apply them often.

In addition, this common interface allows a program written around one device, let's say an NH Research DC Source, to run on a system with another type of device (e.g., an HP DC Source.)

1.3 How to Query Interfaces in Visual Basic

There are two ways to change interface references within Visual Basic. From within the same component

```
Dim obInput as New NHRInputContainer      ' Create a new Input Container
Dim obContainer as INHRContainer          ' Create a pointer to interface
Set obContainer = obInput                 ' Obtain pointer to other interface

                                          ' Do something useful

Set obContainer = Nothing                 ' Release reference to object
Set obInput = Nothing                     ' Release reference to object
```

Get one object from another

```
Dim obDataMgr as New NHRData              ' Create a new Data Manager
Dim obBlobMgr as INHRBlob                 ' Create a pointer to Blob Manager
Set obBlobMgr = obDataMgr.BlobMgr         ' Obtain pointer to Blob Manager
.
.                                         ' Do something useful
.
Set obBlobMgr = Nothing                   ' Release reference to object
Set obDataMgr = Nothing                   ' Release reference to object
```

2. QUICK DISCUSSION OF *em**Power*** ® COMPONENTS

2.1 Description/Use

The *Blob Manager*

The *Blob Manager* performs persistent storage of structured data. Stored data is in the form of a structure (VB = user defined type). We call this structured data a "blob" (*b*inary *l*arge *ob*ject). All data used throughout the system, whether test program, device configuration, or test routine parameters are stored as blobs. The blob manager is like a hatcheck clerk. That is, if a component wants to store some data, it gives that data to the blob manager and the blob manager gives the application a claim-check (or *handle*) that can be used to retrieve it later. Since all system data is stored in these blobs, the blob manager can store and read it back from the disk. The stored blobs *are* the test program.

The *Configuration Server*

The *Configuration Server* controls tester configuration and provides access to tester device objects. When this component loads, it loads all configured device components into memory. It then makes those devices available to anyone who wants them by using a logical name. The Configuration Server also provides a visual editor where the system configuration may be maintained.

The *Datalog Server*

The *Datalog Server* performs datalogging of test results. Test routines log data to the Datalog Server, and the server, in turn, sends the results to each of the Datalog Drivers configured in the test program.

Do not confuse the Datalog Server with the Datalog Drivers – while the drivers, such as the Printer, Screen or Text Driver, perform the actual logging of test results to the appropriate media, the Datalog Server is responsible for controlling the drivers. Some media types take longer to access than others; we let the Datalog Server handle the complexity of managing the Drivers. Remember, test routines log to the Server, never the Drivers directly.

The *Test Program Data Manager* (Data Manager)

The *Test Program Data Manager*, or simply the *Data Manager*, creates and manages all data related to the Test Program. One of the more complicated *em**Power*** ® components, the Data Manger exposes nine different interfaces to control access to various sections of a test program. These interfaces correspond closely to the tabs of the PowerEdit screen – Program, Output, Input, Digital, Analog, and Measurement, along with others necessary for program creation.

The Data Manager creates and maintains the Input, Output and Digital Containers used by test routines to control many of the tester devices, as well as the UUT Communications objects.

The *Fault Manager*

The *Fault Manager* performs centralized fault logging and reporting. If an error is detected, and the component cannot resolve the condition, the error code and any contextual information are logged to the Fault Manager. An application can then ask the Fault Manager for any faults that have occurred and report them to the user though either the Datalog functions or a Windows dialog.

UUT Communication Servers – (Serial, GPIB, CAN, I^2C bus)

Many of today's UUTs contain microcontrollers to provide status and programmability features for both end-user and manufacturing use. *em**Power*** provides an assortment of communication servers to aid in using these capabilities. The servers provide communications both to and from the UUT, verification of returned strings and values, and advanced features such as tokenizing and other string manipulations.

Containers – Output (Load), Input (Source) and Digital device wrappers

As the name suggests, *Containers* are used to wrap, or contain, groups of devices and make them appear as a single unit. Provides three types of Containers – Output Containers, which are the basic control for system Loads, Input Containers, which are the basic control for system Sources, and Digital Containers which control DOUTs and relays.

The Containers simplify the programming of the system Loads, Sources and Digitals. For example, they manage the complexity of attaching more than one load to a UUT output; they can expose a group of Digitals in a bus format.

In addition, Containers are used to describe the wiring connections to the software. They indicate which multiplexer channels are connected to the UUT, provide scale factor information should a current or potential transformer be attached to a channel and tell the system which relay should be controlled when performing an overvoltage test.

The *Test Information* Object

When a test is run on a UUT it is likely that certain information describing this UUT will need to be logged along with the test results. Serial Number, Lot Code, and other user-driven data entry can be performed using the *Test Information* component. It is fully configurable from within PowerEdit, allowing the definition of the data entry fields, the length of the contents, making some fields mandatory while others are optional, auto-increment, etc.

The *Helper* Object

The *Helper* object provides miscellaneous functions not specific to one type of component or another. This component is used to evaluate test routine results, set vectors and control the front panel lights.

The *SPC Server*

The *SPC Server* allows you to monitor your manufacturing process in real-time. Because it works in real-time, the *SPC Server* does not keep historical *SPC* data – rather it only keeps *SPC* data from the recent past, giving you a real-time, representative view of your manufacturing process.

The *SPC Server* has a built-in report generator designed to give you meaningful feedback on you manufacturing process, not the performance of your UUTs – that's what the *Datalog Server* is for.

Test Routines

This is where the real work is done. The test routines provide the algorithms used to test your UUT. While they can take advantage of everything *em**Power*** ® provides, the true success or failure of the test to uncover defects is the responsibility of the test routine.

Device Drivers

Device Drivers are components that control the test system hardware. Each piece of test equipment in the test system requires an appropriate *em**Power*** ® driver in order to be controlled in the software environment. The device drivers are named using a sequential numerical scheme not related to the type of device. For example NHRDev1 is the driver for the NH Research AC/DC Source; NHRDev2 controls the NHR AC Load; NHRDev9 is the Agilent 34401A DMM, etc. As new device drivers are added to the system they are assigned the next available sequential number.

Some pieces of test equipment provide more than a single function to the tester. NHRDev3, the NH Research Digital Measurement System, provides DMM measurement capability, transient analysis, timing measurements, digital input and outputs, general-purpose relays, and more. While one driver controls this multiple function instrument, it has been separated into many logical devices to ease user programming. The user sees distinct "devices", and is unconcerned that they are all the same device.

Because of this, device drivers have a logical ID associated with each capability. Programmers must provide the correct logical ID whenever controlling the device. The correct logical ID is automatically obtained from the *Configuration Server* at the same time that the device reference is obtained.

3. MULTI-PASS REGULATION TEST ROUTINE SAMPLE

Introduction

In this section we will be following a step-by-step procedure to create a test routine. We'll start with the *Test Routine Template*, modified to the step that reads, "You are now ready to modify the test program as necessary to perform your specific task".

Our test routine will be a multiple-pass regulation test. It should allow the user to take a measurement of the UUT in up to five different combinations of *Input* and *Output Vectors*. The algorithm will be to simply apply the vector selections, take the measurement and log the results for each enabled loop, or pass.

Construct the User Interface

The first step will be to create the *User Interface* for our test routine. You can format the display any way you like, but the instructions that follow will assume the following controls and control names:

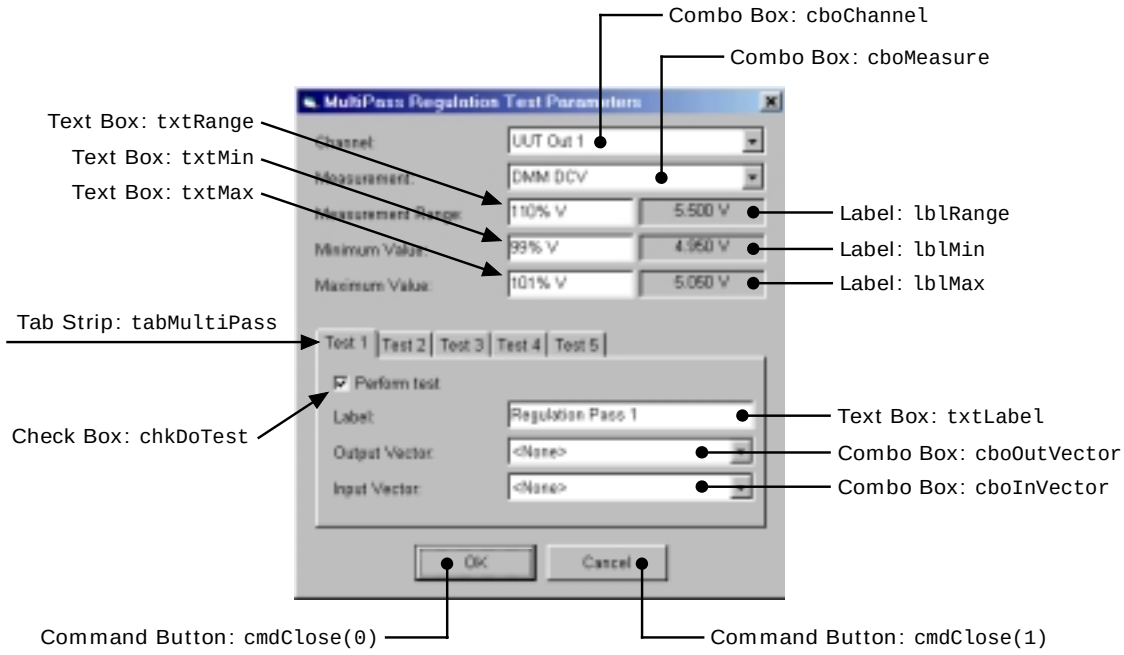

Figure 4 – The *MultiPass* Form Controls (`frmUIProp.frm`)

Add *Output Containers* to `Form_Load()`

Once the form has been designed we'll start to add code. Begin by removing the existing code from the `Form_Load` function. We'll call a subroutine from the *TestHelper* module that fills a *Combo Box* with a list of all the output channels currently configured in the test program. The *Combo Box* will also contain an ID for each entry in the corresponding *ItemData* field. This will be useful later on when we wish to refer to the selected *Container* again. This routine is found in `frmUIProp.frm`

```
Private Sub Form_Load()
    AddOutputContainers cboChannel
End Sub
```

Create `cboChannel_Click()` and add the function `AddMeasVectors()`

Whenever the user changes the active channel, the list of applicable measurement vectors changes as well. This is because we want to expose measurement vectors that are specific to this selected output channel in addition to generic DMM-type measurement vectors. We'll call another subroutine from the *TestHelper* module which fills a combo box with a list of all the currently defined measurement vectors which meet our criterion – in this case, a mixture of vectors made by the selected *Output Container* and those made by generic DMM-type measurement devices. This routine is found in `frmUIProp.frm`

```
Private Sub cboChannel_Click()
    'Whenever the user changes the active channel, we have to get an updated list
    'of measurement vectors applicable to the new channel
    AddMeasVectors cboMeasure, OUTPUT_CONTAINER,
    cboChannel.List(cboChannel.ListIndex), "DMM"
End Sub
```

Add `SetReferences()` to `cboChannel_Click()`, and create the function itself

In order to support referential values (those entered using percentages) we need to know the *Reference Values* for the selected *Output Container*. To accomplish this, we must edit three areas of code.

We'll start by creating some module-level variables to hold the values of the various references exposed by the *Output Containers*. We will be editing various areas of `frmUIProp.frm`

```
Option Explicit
Dim dRefVoltage As Double
Dim dRefCurrent As Double
Dim dRefResistance As Double
Dim dRefWattage As Double
Dim dRefFrequency As Double
```

Next, we'll add the code that gathers the reference values from the active Output Container. This object is created by the *Data Manager*, and is accessible by ID via the *INHRDataOutput* interface. The *Container ID* was stored in the *ItemData* element of the *Channel Combo Box* by the helper function `AddOutputContainers()` (*See:* Add *Output Containers* to **`Form_Load()`**).

Passing this ID to the *Data Manager* via `GetOutContainerObj()` will retrieve the *Container Object*. We "set" the returned object into an *NHROutputContainer* variable to gain

access to the *Output Container* interface. From there we simply access each *Reference Property* and copy the values into our module-level variables.

```
Sub SetReferences()
    Dim hID As Long
    Dim obDataMgrOut As INHRDataOutput
    Dim obOutContainer As NHROutputContainer

    'Get the Container ID from the ComboBox item data
    hID = cboChannel.ItemData(cboChannel.ListIndex)

    'Get a pointer to the Output interface of the Data Manager, then get the
    'appropriate Output Container (based on it's ID)
    Set obDataMgrOut = g_obDataMgr
    Set obOutContainer = obDataMgrOut.GetOutContainerObj(hID)

    'Update our reference values
    dRefVoltage = obOutContainer.RefVoltage
    dRefCurrent = obOutContainer.RefCurrent
    dRefResistance = obOutContainer.RefResistance
    dRefWattage = obOutContainer.RefWatts
    dRefFrequency = obOutContainer.RefFrequency

    Set obOutContainer = Nothing
    Set obDataMgrOut = Nothing
End Sub
```

We'll finish by adding the `SetReferences()` call to the `cboChannel_Click()` event. This way we get new reference values every time the user sets a new active channel.

```
Private Sub cboChannel_Click()
    'Whenever the user changes the active channel, we have to get an updated list
    'of measurement vectors applicable to the new channel
    AddMeasVectors cboMeasure, OUTPUT_CONTAINER, cboChannel.List(cboChannel.ListIndex)

    'Get the reference values for the new channel
    SetReferences
End Sub
```

Create `cboMeasure_Click()`

We'll take advantage of a few more *Helper* functions when dealing with measurements. Most of these calls want to know the units ("V", "A", "W") of the selected measurement vector. Set up another module-level variable to hold the unit string. We will be editing various areas of `frmUIProp.frm`

```
Option Explicit
Dim dRefVoltage As Double
Dim dRefCurrent As Double
Dim dRefResistance As Double
Dim dRefWattage As Double
Dim dRefFrequency As Double
Dim g_sUnits As String
```

The unit descriptor needs to change each time the user selects a different measurement vector. Since our measurement vector *Combo Box* contains a mixture of generic DMM-type measurements and those specific to the active *Output Container* (*See:* Create **`cboChannel_Click()`** and add the function **`AddMeasVectors()`**), we need to determine which object is performing the measurement. Fortunately, we have another *Helper* function to perform this task for us.

The function requires the logical name of the device exposing the selected measurement vector, a pointer to the active *Output Container*, and the handle to blob defining the measurement.

```
Private Sub cboMeasure_Click()
    Dim hID As Long
    Dim obDataMgrOut As INHRDataOutput
    Dim obContainer As INHRContainer
    Dim obDataMgrMeas As INHRDataMeas
    Dim sDevName As String
    Dim hBlob As Long

    'Get the Container ID from the ComboBox item data
    hID = cboChannel.ItemData(cboChannel.ListIndex)

    'Get a pointer to the Output interface of the Data Manager, then get the
    'appropriate Output Container (based on it's ID)
    Set obDataMgrOut = g_obDataMgr
    Set obContainer = obDataMgrOut.GetOutContainerObj(hID)
    Set obDataMgrOut = Nothing

    'Get a pointer to the Measurement interface of the Data Manager, then get the
    'the name of the device making the measurement and the blob which defines the
    'selected measurement vector.  Once we have this information, we can ask for
    'the measurement units.
    Set obDataMgrMeas = g_obDataMgr
    sDevName = obDataMgrMeas.GetMeasDevName(cboMeasure.ItemData(cboMeasure.ListIndex))
    hBlob = obDataMgrMeas.GetMeasDevProp(cboMeasure.ItemData(cboMeasure.ListIndex))
    g_sUnits = GetUnits(sDevName, obContainer, hBlob)

    Set obDataMgrMeas = Nothing
    Set obContainer = Nothing
End Sub
```

Create `GotFocus()` and `LostFocus()` events for `txtRange`, `txtMin` and `txtMax`, and the `GotFocus()` event for `txtLabel`

It's important to present a consistent data entry experience to the user. When the cursor enters a text field, we do them the courtesy of pre-selecting the existing text so they don't have to delete it themselves. Likewise, when the user leaves the field, we want to format the data in a consistent way.

In the `GotFocus()` event for each of the *Text Box* controls we'll call a simple *Helper* function designed to highlight any existing text in the provided control.

In each `LostFocus()` event, we'll call a formatting function provided with the *Engineering Value* module `EngValue.bas` to translate the entry into an *Engineering Format* while appending the appropriate measurement unit specifier to the text. These routines are found in `frmUIProp.frm`

```
Private Sub txtLabel_GotFocus()
    ' Pre-select the existing text in the control so the user doesn't have to
    SelectText txtLabel
End Sub

Private Sub txtMax_GotFocus()
    ' Pre-select the existing text in the control so the user doesn't have to
    SelectText txtMax
End Sub

Private Sub txtMax_LostFocus()
    ' Add the correct units to whatever text the user entered
    txtMax.Text = AddUnits(txtMax.Text, g_sUnits)
End Sub

Private Sub txtMin_GotFocus()
    ' Pre-select the existing text in the control so the user doesn't have to
    SelectText txtMin
End Sub
```

```
Private Sub txtMin_LostFocus()
    ' Add the correct units to whatever text the user entered
    txtMin.Text = AddUnits(txtMin.Text, g_sUnits)
End Sub

Private Sub txtRange_GotFocus()
    ' Pre-select the existing text in the control so the user doesn't have to
    SelectText txtRange
End Sub

Private Sub txtRange_LostFocus()
    ' Add the correct units to whatever text the user entered
    txtRange.Text = AddUnits(txtRange.Text, g_sUnits)
End Sub
```

Update the contents of the "*Text Reference Helper*" fields in the `LostFocus()` events for `txtRange`, `txtMin`, and `txtMax`

Part of the consistent data entry experience is provided through the use of the "reference helper" labels located next to the text fields. When the user enters a data value as a percentage of reference, this area displays the actual data value that will be used by the test routine.

There is a *Helper* function named `CalculateReference()` that calculates, and formats this reference helper value for us; it requires the correct reference value (remember Add **`SetReferences()`** to **`cboChannel_Click()`**, and create the function itself – we have a few possibilities), the *Label* control to display the resulting text, the user-entered text string to convert, the units string, and two additional fields for handling values which are offsets from a nominal value. For our application we won't be using either of the offset parameters so we'll set them to "False". These routines are found in `frmUIProp.frm`

```
Sub DoReferenceCalc(lbl As Label, sText As String)
    Dim dRef As Double

    ' Determine the correct reference according to the units of the provided text
    If InStr(g_sUnits, "VA") Or InStr(g_sUnits, "W") Then
        dRef = dRefWattage
    ElseIf InStr(g_sUnits, "V") Then
        dRef = dRefVoltage
    ElseIf InStr(g_sUnits, "A") Then
        dRef = dRefCurrent
    ElseIf InStr(g_sUnits, "O") Or InStr(g_sUnits, "R") Then
        dRef = dRefResistance
    ElseIf InStr(g_sUnits, "H") Then
        dRef = dRefFrequency
    End If
    ' Call helper function to fill in the reference field correctly
    CalculateReference dRef, lbl, sText, False, g_sUnits, False
End Sub
```

In each `LostFocus()` event, we'll call our `DoReferenceCalc()` formatting function.

```
Private Sub txtMax_LostFocus()
    ' Add the correct units to whatever text the user entered,
    ' and update the "helper" fields.
    txtMax.Text = AddUnits(txtMax.Text, g_sUnits)
    DoReferenceCalc lblMax, txtMax.Text
End Sub

Private Sub txtMin_LostFocus()
    ' Add the correct units to whatever text the user entered,
    ' and update the "helper" fields.
    txtMin.Text = AddUnits(txtMin.Text, g_sUnits)
    DoReferenceCalc lblMin, txtMin.Text
End Sub
```

```
Private Sub txtRange_LostFocus()
    ' Add the correct units to whatever text the user entered,
    ' and update the "helper" fields.
    txtRange.Text = AddUnits(txtRange.Text, g_sUnits)
    DoReferenceCalc lblRange, txtRange.Text
End Sub
```

Add direct calls to the `LostFocus()` events in `cboChannel_Click()` and `cboMeasure_Click()`

Now that we have a method of updating the text fields and associated helper labels we need to insure they remain up to date while the user interacts with the form.

When a measurement vector selection changes, we have the potential of the units changing, which may change the selected reference value. By directly calling the `LostFocus()` events of our text boxes inside the *Click* event of our measurement *Combo Box* both the text and helper labels will get updated with any changes. These routines are found in `frmUIProp.frm`

```
Private Sub cboMeasure_Click()
    Dim hID As Long
    Dim obDataMgrOut As INHRDataOutput
    Dim obContainer As INHRContainer
    Dim obDataMgrMeas As INHRDataMeas
    Dim sDevName As String
    Dim hBlob As Long

    'Get the Container ID from the ComboBox item data
    hID = cboChannel.ItemData(cboChannel.ListIndex)

    'Get a pointer to the Output interface of the Data Manager, then get the
    'appropriate Output Container (based on it's ID)
    Set obDataMgrOut = g_obDataMgr
    Set obContainer = obDataMgrOut.GetOutContainerObj(hID)
    Set obDataMgrOut = Nothing

    'Get a pointer to the Measurement interface of the Data Manager, then get the
    'the name of the device making the measurement and the blob which defines the
    'selected measurement vector.  Once we have this information, we can ask for
    'the measurement units.
    Set obDataMgrMeas = g_obDataMgr
    sDevName = obDataMgrMeas.GetMeasDevName(cboMeasure.ItemData(cboMeasure.ListIndex))
    hBlob = obDataMgrMeas.GetMeasDevProp(cboMeasure.ItemData(cboMeasure.ListIndex))
    g_sUnits = GetUnits(sDevName, obContainer, hBlob)

    'Update "helper" boxes to reflect the new reference values
    txtRange_LostFocus
    txtMin_LostFocus
    txtMax_LostFocus

    Set obDataMgrMeas = Nothing
    Set obContainer = Nothing
End Sub
```

We also need to call those same events in the channel *Click* event in case the user changes the active channel because the reference values may have changed.

```
Private Sub cboChannel_Click()
    'Whenever the user changes the active channel, we have to get an updated list
    'of measurement vectors applicable to the new channel
    AddMeasVectors cboMeasure, OUTPUT_CONTAINER, cboChannel.List(cboChannel.ListIndex)

    'Get the reference values for the new channel
    SetReferences

    'Update "helper" boxes to reflect the new reference values
    txtRange_LostFocus
```

```
        txtMin_LostFocus
        txtMax_LostFocus
    End Sub
```

Create the `chkDoTest_Click()` event

Another aspect of the consistent data entry experience occurs when features of the user interface gray-out and become disabled when they are not applicable. In this case we'll disable the controls on the *Tab Strip* when the *Perform test* check box is cleared, and enable them when the check box is checked.

It's best to use system colors such as *WindowBackground* and *ButtonFace* instead of palette colors so the forms display well in each user's color scheme. This routine is found in `frmUIProp.frm`

```
    Private Sub chkDoTest_Click()
        Dim bEnabled As Boolean
        Dim lBackColor As Long

        If chkDoTest.Value = vbChecked Then
            bEnabled = True
            lBackColor = vbWindowBackground
        Else
            bEnabled = False
            lBackColor = vbButtonFace
        End If

        txtLabel.Enabled = bEnabled
        txtLabel.BackColor = lBackColor
        cboInVector.Enabled = bEnabled
        cboInVector.BackColor = lBackColor
        cboOutVector.Enabled = bEnabled
        cboOutVector.BackColor = lBackColor
    End Sub
```

Create `tabMultiPass_Click()`

Unlike some third-party tab controls, the *Tab Strip* provided with *Visual Basic* does not automatically hide controls placed on a tab and show a different set of controls when the active tab is changed. However, our user interface is designed to take advantage of this behavior. Since we're not swapping out the controls whenever the active tab is changed, we must swap out the *contents* of the controls.

The first thing we want to do is create a module-level variable to hold the index of the currently selected tab. We will be editing various areas of `frmUIProp.frm`

```
    Option Explicit
    Dim dRefVoltage As Double
    Dim dRefCurrent As Double
    Dim dRefResistance As Double
    Dim dRefWattage As Double
    Dim dRefFrequency As Double
    Dim g_sUnits As String
    Dim g_lActiveTab As Long
```

When the user clicks on a tab we want to save away the contents of the controls for the originally selected tab and then redraw the controls with data reflecting the properties of the new tab index. Later on we'll create a subroutine to save the properties `SaveProperties()` and another to

update the controls with the new property data `DisplayProperties()`; for now, simply enter the subroutine names in the proper code sequence. (Note: it is important to save the current properties *before* updating the module-level "active tab" variable and to display the new properties *after* updating the "active tab" variable.)

```
Private Sub tabMultiPass_Click()
    ' Whenever we change tabs, first save away the current contents of the controls
    ' on the tab strip, then change the active tab indicator, and finally update the
    ' controls with appropriate settings for the new active tab.
    SaveProperties
    g_lActiveTab = tabMultiPass.SelectedItem.Index
    DisplayProperties
    chkDoTest_Click
End Sub
```

Modify the *Blob* definition

Before we can save the contents of the controls, we need a structure to hold the data. In *emPower*®, this structure is known as a *Property Blob*. There should be an entry in the property blob for each control on the form.

Since we refer to the *Output Container* and *Measurement Vectors* by ID (*See:* Add *Output Containers* to **`Form_Load()`**, and Create **`cboMeasure_Click()`**), we should store those IDs in the *Blob*. The *Range*, *Minimum*, and *Maximum Value* parameters are stored as 20-character fixed length strings. We choose strings over doubles so the user can enter values such as "99.5% V". (Note: any strings stored in blobs must be fixed length).

The remaining entries are stored as arrays of five elements to correspond to the five regulation steps that our test routine is able to perform. We need to save the state of the check box, the descriptive text (32-character fixed length strings) and the *Input* and *Output Vector* selections, stored by ID. Begin by removing the existing code in this function. We will be adding this structure to `NHRTRMultiPass.bas`

```
'MultiPass test properties
Public Type MultiPassProp
    hID As Long                         'Container ID
    lMeasureVectorID As Long            'Measurement vector ID
    sRange As String * iMAXNORMVALLEN   'Measurement range
    sMin As String * iMAXNORMVALLEN     'Minimum tolerance
    sMax As String * iMAXNORMVALLEN     'Maximum tolerance

    bDoTest(1 To 5) As Long                     'TRUE to perform test
    sLabel(1 To 5) As String * iMAXNORMNAMELEN  'Descriptive name of this test setup
    lInputVectorID(1 To 5) As Long              'Measurement vector ID
    lOutputVectorID(1 To 5) As Long             'Measurement vector ID
End Type
```

Create *SaveProperties*

Now that we have a property blob, we can use it to save the selections the user entered via the user interface. When saving the array-based entries, we use the module-level "active tab" variable to determine which elements to save. This routine belongs in `frmUIProp.frm`

```
Sub SaveProperties()
    With g_blobProp
        If cboChannel.ListIndex <> -1 Then
            .hID = cboChannel.ItemData(cboChannel.ListIndex)
        End If
```

```
            If cboMeasure.ListIndex <> -1 Then
                .lMeasureVectorID = cboMeasure.ItemData(cboMeasure.ListIndex)
            End If

            .sRange = Trim$(txtRange.Text)
            .sMin = Trim$(txtMin.Text)
            .sMax = Trim$(txtMax.Text)

            If g_lActiveTab <> 0 Then
                .bDoTest(g_lActiveTab) = chkDoTest.Value
                .sLabel(g_lActiveTab) = txtLabel.Text

                If cboInVector.ListIndex <> -1 Then
                    .lInputVectorID(g_lActiveTab) =
                    cboInVector.ItemData(cboInVector.ListIndex)
                End If

                If cboOutVector.ListIndex <> -1 Then
                    .lOutputVectorID(g_lActiveTab) =
                    cboOutVector.ItemData(cboOutVector.ListIndex)
                End If
            End If
        End With
    End Sub
```

Create *DisplayProperties*

When the user changes the active tab we only need to update the controls that reside on the *Tab Strip*. For the *Check Box* and the *Text Label* we'll transfer data directly from the appropriate blob elements, indexing the arrays according to the active tab value. We trim the text coming from the blob because the fixed length strings transfer whitespace into the text fields.

We'll use the `FindVectorInList()` *Helper* subroutine to find the correct entry in the *Input* and *Output Container* combo boxes. You simply pass it the *Combo Box* containing the vector entries and the ID of the desired vector and it will select the correct entry automatically. This routine belongs in `frmUIProp.frm`

```
    Sub DisplayProperties()
        With g_blobProp
            chkDoTest.Value = .bDoTest(g_lActiveTab)
            txtLabel.Text = Trim$(.sLabel(g_lActiveTab))
            FindVectorInList cboInVector, .lInputVectorID(g_lActiveTab)
            FindVectorInList cboOutVector, .lOutputVectorID(g_lActiveTab)
        End With
    End Sub
```

Setup initial states during `Form_Load()`

It is normal to perform some initialization of the controls on the form before it is visible. We've already loaded the *Channel Combo Box* with a list of available *Output Containers*; we now need to initialize the *Input* and *Output Vector Combo Boxes*. We'll use a pair of *Helper* functions to assist us in this. The `AddInVectors()` and `AddOutVectors()` subroutines take the supplied combo box and fill it in with the available *Input* and *Output Vectors*, respectively. Like `AddOutputContainers()`, the *ItemData* field for each entry is loaded with the ID of each vector. The second parameter determines whether a "<None>" entry exists in the list. We'll set this parameter to *True* to include the "<None>" entry to allow the user to tell us they do not wish to change the state of the *Input* or *Output Container*.

While we have to explicitly set the values of the controls on the form (*Channel* and *Measurement Combo Boxes*, *Range*, *Min* and *Max Text Boxes*), we can call functions that we've already written to initialize the controls on the *Tab Strip*. This routine is found in `frmUIProp.frm`

```
Private Sub Form_Load()
    AddOutputContainers cboChannel
    AddInVectors cboInVector, True
    AddOutVectors cboOutVector, True

    With g_blobProp
        FindVectorInList cboChannel, .hID
        FindVectorInList cboMeasure, .lMeasureVectorID
        txtRange.Text = Trim$(.sRange)
        txtMin.Text = Trim$(.sMin)
        txtMax.Text = Trim$(.sMax)
    End With

    g_lActiveTab = 1
    DisplayProperties
    cboMeasure_Click
End Sub
```

Add code to `InitializePropBlob()`

The template provides a subroutine used to initialize the property blob structure. Visual Basic will ensure that the numeric fields are zero and that the strings are empty but this may not be the initial states we desire. Of particular note here is the initial state of the boolean entries – since we transfer the value of the property blob directly into the value property of the check box we want to use the constants `vbChecked`, and `vbUnchecked` instead of *True* and *False*. Begin by removing the existing code in this function. This routine is found in `NHRTRMultiPass.bas`

```
Sub InitializePropBlob()
    Dim i As Integer

    With g_blobProp
        .hID = 0
        .lMeasureVectorID = 0
        .sRange = "110%"
        .sMin = "99%"
        .sMax = "101%"

        For i = 1 To 5
            .bDoTest(i) = vbUnchecked   ' Use Unchecked instead of False
            .sLabel(i) = "Regulation Pass " & i
            .lInputVectorID(i) = 0
            .lOutputVectorID(i) = 0
        Next

        .bDoTest(1) = vbChecked     ' Start with the first test enabled
    End With
End Sub
```

Add call to `InitializePropBlob()` in `INHRTR_Init()`

Now that we have a way to initialize the property blob, we need to determine where is the correct place to call the subroutine. Normally, ***Power****EDIT* will call `LoadBlobProp()` to load a specific property blob prior to calling `UIProp()` to display the user interface (the exception is the first time a test routine is accessed after insertion into a test program). If we were to place `InitializePropBlob()` in `Form_Load()` we would end up replacing the properties

loaded by ***Power**EDIT* with the initialized states. The call to Init is made immediately after the test routine is created (before `LoadBlobProp()`); this is a good place to initialize our property blob. . This routine is found in `NHRTRMultiPass.cls`

```
Private Sub INHRTR_Init(ByVal hwndParent As Long, _
                        ByVal pdispDatalogServer As Object, _
                        ByVal pdispDataManager As Object, _
                        ByVal pdispCfgManager As Object, _
                        ByVal pdispHelperSvr As Object, _
                        ByVal pdispFaultMgr As Object, _
                        ByVal pdispTestInfoSvr As Object)

    g_hwndParent = hwndParent
    Set g_obDatalog = pdispDatalogServer
    Set g_obDataMgr = pdispDataManager
    Set g_obCfgMgr = pdispCfgManager
    Set g_obHelper = pdispHelperSvr
    Set g_obFaultMgr = pdispFaultMgr
    Set g_obTestInfo = pdispTestInfoSvr
    InitializePropBlob
End Sub
```

Add Printer support

***Power**EDIT* relies on our test routine object to provide two different print styles. The first is a vertically favored full format. Every property must be translated in a human readable format. The data should be complete enough to enable a programmer to recreate the test step from the printout. We call this full format the *RegularDump*. These routines are found in `NHRTRMultiPass.bas`

```
Function RegularDump(bPrinterFormat As Long) As String
    Dim sDump As String
    Dim sIndent As String
    Dim sIndent2 As String
    Dim sDesc As String
    Dim i As Integer
    Dim obDataMgrMeas As INHRDataMeas
    Dim obDataMgrOut As INHRDataOutput
    Dim obDataMgrIn As INHRDataInput
    Dim obContainer As INHRContainer

    On Error GoTo INHRTR_DumpPropError

    Set obDataMgrMeas = g_obDataMgr
    Set obDataMgrOut = g_obDataMgr
    Set obDataMgrIn = g_obDataMgr

    sIndent = GetIndent(bPrinterFormat)
    sIndent2 = GetIndent(bPrinterFormat + 1)

    sDump = sIndent & sNHRTRName & " Configuration" & vbCrLf

    With g_blobProp
        Set obContainer = obDataMgrOut.GetOutContainerObj(.hID)
        sDump = sDump & sIndent & "Channel: " & RTrim$(obContainer.Describe) & vbCrLf
        sDump = sDump & sIndent & "Measurement: " &
        obDataMgrMeas.GetMeasVectorLabel(.lMeasureVectorID) & vbCrLf
        sDump = sDump & sIndent & "Measurement Range: " & RTrim$(.sRange) & vbCrLf
        sDump = sDump & sIndent & "Minimum Value: " & RTrim$(.sMin) & vbCrLf
        sDump = sDump & sIndent & "Maximum Value: " & RTrim$(.sMax) & vbCrLf

        For i = 1 To 5
            sDump = sDump & vbCrLf
            sDump = sDump & sIndent & "Test " & i & vbCrLf
            sDump = sDump & sIndent2 & "Perform test: " & Format(.bDoTest(i),
            "True/False") & vbCrLf
            If .bDoTest(i) Then
```

```
                sDump = sDump & sIndent2 & "Label:" & .sLabel(i) & vbCrLf
                If .lOutputVectorID(i) <> 0 Then
                    sDesc = obDataMgrOut.GetOutVectorLabel(.lOutputVectorID(i))
                Else
                    sDesc = "<None>"
                End If
                sDump = sDump & sIndent2 & "Output Vector:" & sDesc & vbCrLf
                If .lInputVectorID(i) <> 0 Then
                    sDesc = obDataMgrIn.GetInVectorLabel(.lInputVectorID(i))
                Else
                    sDesc = "<None>"
                End If
                    sDump = sDump & sIndent2 & "Input Vector:" & sDesc & vbCrLf
            End If
        Next
    End With
    DumpPropExit:
    Set obContainer = Nothing
    Set obDataMgrIn = Nothing
    Set obDataMgrOut = Nothing
    Set obDataMgrMeas = Nothing

    RegularDump = sDump
    Exit Function

    INHRTR_DumpPropError:
    Dim sMsg As String
    sMsg = "Unable to dump property blob: " & Err.Description
    MsgBox sMsg, vbCritical Or MB_TOPMOST
    sDump = sDump & sMsg
    Resume DumpPropExit
End Function
```

The second style is a horizontally favored summary format. The purpose of this format is to provide enough information to give the programmer an idea of what is being tested, without being overwhelmed with minute details. We call this format the *SummaryDump*.

```
Function SummaryDump(bPrinterFormat As Long) As String
    Dim sDump As String
    Dim sDesc As String
    Dim i As Integer
    Dim obDataMgrMeas As INHRDataMeas
    Dim obDataMgrOut As INHRDataOutput
    Dim obDataMgrIn As INHRDataInput
    Dim obContainer As INHRContainer

    On Error GoTo INHRTR_DumpPropError

    Set obDataMgrMeas = g_obDataMgr
    Set obDataMgrOut = g_obDataMgr
    Set obDataMgrIn = g_obDataMgr

    For i = 1 To Len(sNHRTRName)
        If Mid(sNHRTRName, i, 1) = ";" Then
            Exit For
        End If
    Next
    sDump = Left(sNHRTRName, i - 1) & vbCrLf

    With g_blobProp
        Set obContainer = obDataMgrOut.GetOutContainerObj(.hID)
        sDump = sDump & vbTab & "Channel: " & RTrim$(obContainer.Describe) & vbTab &
        vbTab
        sDump = sDump & "Measurement: " &
        obDataMgrMeas.GetMeasVectorLabel(.lMeasureVectorID) & vbTab & vbTab
        sDump = sDump & "Minimum Value: " & RTrim$(.sMin) & vbTab & vbTab
        sDump = sDump & "Maximum Value: " & RTrim$(.sMax) & vbCrLf

        sDump = sDump & vbTab & "Selected tests:" & vbCrLf
        For i = 1 To 5
            If .bDoTest(i) Then
```

```
                sDump = sDump & vbTab & vbTab & i & ".) " & .sLabel(i) & vbTab & vbTab
                If .lOutputVectorID(i) <> 0 Then
                    sDesc = obDataMgrOut.GetOutVectorLabel(.lOutputVectorID(i))
                Else
                    sDesc = "<None>"
                End If
                sDump = sDump & "Output Vector:" & sDesc & vbTab & vbTab
                If .lInputVectorID(i) <> 0 Then
                    sDesc = obDataMgrIn.GetInVectorLabel(.lInputVectorID(i))
                Else
                    sDesc = "<None>"
                End If
                sDump = sDump & "Input Vector:" & sDesc & vbCrLf
            End If
        Next
    End With

    DumpPropExit:
    Set obContainer = Nothing
    Set obDataMgrIn = Nothing
    Set obDataMgrOut = Nothing
    Set obDataMgrMeas = Nothing
    SummaryDump = sDump
    Exit Function

    INHRTR_DumpPropError:
    Dim sMsg As String
    sMsg = "Unable to dump property blob: " & Err.Description
    MsgBox sMsg, vbCritical Or MB_TOPMOST
    sDump = sDump & sMsg
    Resume DumpPropExit
End Function
```

Comment out the body of `INHRTR_Go()`, and test interface functionality

The template contains code in the `Go()` function that will no longer compile (due to the property blob change). At this point let's comment out the entire contents of the function and test our user interface functionality. The `Go()` routine is found in `NHRTRMultiPass.cls`

1. The user should not be able to enter any data that would cause the user interface to crash
2. The initial state of the form controls should be reasonable
3. Changing the *Channel* selection should cause the percentage reference values in the label fields to update
4. Changing the *Measurement* selection should cause the units in the text and label fields to update
5. Toggling the state of the check box should enable and disable the controls on the tab strip
6. Gaining focus in the text boxes should pre-select the existing text
7. Losing focus from the text boxes should reformat the contents of both the text box and the corresponding label
8. Changing the active tab should cause the controls on the tab strip to update accordingly

9. The *ToolTip* help messages should be reasonable

10. If *OK* is pressed any changes made to the contents of the controls should be saved and restored when the test routine is reloaded

11. If *Cancel* is pressed any changes made to the contents of the controls should be discarded and the previous contents should be restored when the routine is reloaded

12. Both the normal and summary print options should be functional and look reasonable

Add the test algorithm to *INHRTR_Go*

Now that our user interface is functioning correctly, we can add the test algorithm. Refer to the comments in the code for explainations of the algorithm. The algorithm belongs in the `Go()` function of `NHRTRMultiPass.cls`

```
Private Function INHRTR_Go(ByVal bDebug As Long, ByVal InitialVectorID As Long, _
                           Optional ByVal bAlwaysFail As Long = 0&) As Long
    Dim iLoop As Integer
    Dim obDmm As INHRDmm
    Dim lLogID As Long
    Dim sMuxChannel As String
    Dim hBlob As Long
    Dim dRange As Double
    Dim sUnits As String
    Dim dReference As Double

    ' Datalog variables
    Dim dMin As Double
    Dim dMax As Double
    Dim dMeasurement As Double
    Dim sResult As String
    Dim lResultType As RESULT_TYPE
    Dim lFailCount As Long

    ' Fault Manager variables
    Dim lFault As Long
    Dim lSecondary As Long
    Dim sDescription As String
    Dim sMoreInfo As String
    Dim dateCode As Date

    On Error GoTo INHRTR_GoError

    lFailCount = SetupMeasurement(obDmm, lLogID, sMuxChannel, hBlob, dRange, _
                                  sUnits, dReference)

    If lFailCount = 0 Then
        With g_blobProp
            ' Determine Min/Max values
            dMin = TranslateValue(.sMin, dReference)
            dMax = TranslateValue(.sMax, dReference)

            For iLoop = 1 To 5
                If .bDoTest(iLoop) Then
                    ' Set the Output Vector, if necessary
                    If .lOutputVectorID(iLoop) <> 0 Then
                        g_obHelper.SetOutputVector .lOutputVectorID(iLoop)
                    End If

                    ' Set the Input Vector, if necessary
                    If .lInputVectorID(iLoop) <> 0 Then
                        g_obHelper.SetInputVector USE_VECTOR_ID, _
                                                  .lInputVectorID(iLoop)
```

```
                    End If

                    ' Take the measurement
                    obDmm.SetupBlobs lLogID, sMuxChannel, hBlob, dRange
                    obDmm.Meas lLogID
                    If obDmm.Fetch(lLogID, dMeasurement) = 0 Then
                        g_obDatalog.Label 0, "Unable to fetch measurement for " _
                                           & Trim$(.sLabel(iLoop)), True, 0, 0
                        lFailCount = lFailCount + 1
                    Else
                        ' Determine Pass/Fail and log measurement
                        lResultType = g_obHelper.EvaluateValResult(DBL_MIN, dMin, _
                                dMeasurement, dMax, DBL_MAX, sResult, bAlwaysFail)
                        If lResultType = FAIL_RESULT Then
                            lFailCount = lFailCount + 1
                        End If
                        g_obDatalog.ValResult 1, Trim$(.sLabel(iLoop)), dMin, _
                                dMeasurement, dMax, sUnits, sResult, lResultType, 0, 0
                    End If
                Else
                    g_obDatalog.ValResult 1, Trim$(.sLabel(iLoop)), 0#, 0#, 0#, _
                            "", "Untested", UNTESTED_RESULT, 0, 0
                End If
            Next
        End With
    End If

    ' Check for faults; log any faults found
    If g_obFaultMgr.CheckFault Then
        lFault = g_obFaultMgr.GetFault(lSecondary, sDescription, sMoreInfo, dateCode)
        g_obFaultMgr.ClearFault
        g_obDatalog.Label 0, "Measurement error: " & sDescription & "; " & _
                        sMoreInfo, True, 0, 0
        lFailCount = lFailCount + 1
    End If

INHRTR_GoEnd:
    INHRTR_Go = lFailCount

    Exit Function

INHRTR_GoError:
    g_obDatalog.Label 0, Err.Description, True, 0, 0
    lFailCount = lFailCount + 1
    Resume INHRTR_GoEnd
End Function
```

Add `SetupMeasurement()`

Here's the code for the function used to setup the measurement parameters. Again, refer to the comments in the code for explainations. This function belongs in `NHRTRMultiPass.bas`

```
Function SetupMeasurement(obDmm As INHRDmm, lLogID As Long, sMuxChannel As String, _
        hBlob As Long, dRange As Double, sUnits As String, dReference As Double _
        ) As Long
    Dim obOutContainer As NHROutputContainer
    Dim obDataMgrOutput As INHRDataOutput
    Dim obDataMgrMeas As INHRDataMeas
    Dim obContainerMux As INHRContainerMux
    Dim sDevName As String
    Dim lFailCount As Long

    ' We're going to need access to various interfaces of the Data Manager...
    Set obDataMgrOutput = g_obDataMgr
    Set obDataMgrMeas = g_obDataMgr

    ' Get the Output Container from the Data Manager based on ID
    Set obOutContainer = obDataMgrOutput.GetOutContainerObj(g_blobProp.hID)
```

```
    ' Get the logical name of the measurement device based on the measurement
    ' vector ID
    sDevName = obDataMgrMeas.GetMeasDevName(g_blobProp.lMeasureVectorID)

    ' Get the handle to the measurement vector property blob
    hBlob = obDataMgrMeas.GetMeasDevProp(g_blobProp.lMeasureVectorID)

    ' Get the measurement multiplexer channel from the Output Container.  The channel
    ' resides in a special Mux interface exposed by the container.  Do NOT use the
    ' "Channel" property exposed in the NHROutputContainer interface as it does not
    ' provide scale factor support.
    Set obContainerMux = obOutContainer
    sMuxChannel = obContainerMux.MuxChannel
    Set obContainerMux = Nothing
    ' Get the measurement units from one of the tolerance entries, and determine the
    ' correct reference value from the units.
    sUnits = Trim$(UCase(ExtractUnits(g_blobProp.sMin)))
    If InStr(sUnits, "VA") Or InStr(sUnits, "W") Then
        dReference = obOutContainer.RefWatts
    ElseIf InStr(sUnits, "A") Then
        dReference = obOutContainer.RefCurrent
    ElseIf InStr(sUnits, "O") Or InStr(sUnits, "R") Then
        dReference = obOutContainer.RefResistance
    ElseIf InStr(sUnits, "H") Then
        dReference = obOutContainer.RefFrequency
    Else
        dReference = obOutContainer.RefVoltage
    End If

    ' Determine the measurement range
    dRange = TranslateValue(g_blobProp.sRange, dReference)

    ' Find the actual measurement device.  If it's not a member of the Configurator,
    ' it must be the active Output Container.
    Set obDmm = g_obCfgMgr.GetLogicalDev(sDevName, lLogID)
    If obDmm Is Nothing Then
        Set obDmm = obOutContainer
    End If
    If obDmm Is Nothing Then
        g_obDatalog.Label 0, "Failed to get DMM device " & sDevName, True, 0, 0
        lFailCount = lFailCount + 1
    End If

    Set obOutContainer = Nothing
    Set obDataMgrOutput = Nothing
    Set obDataMgrMeas = Nothing

    SetupMeasurement = lFailCount
End Function
```

4. AN *emPower*® COM OVERVIEW

Table 1 – *emPower*® Components

Description	File	Class	Interfaces	ProgID	Categories
Blob Manager	NHRBlobMgr.dll	NHRBlob	INHRBlob	NHRBlobMgr.NHRBlob	Automation Objects NHR Component
Configuration Server	NHRCfgSvr.dll	NHRCfg	INHRCfg	NHRCfgMgr.NHRCfg	Automation Objects NHR Component NHR Configuration
Datalog Server	NHRDatalogSvr.exe	NHRDatalog	INHRDatalog	NHRDatalogSvr.NHRDatalog	Automation Objects NHR Component
Test Program Data Manager	NHRDataMgr.dll	NHRData	INHRData INHRDataAnalog INHRDataDatalog INHRDataDigital INHRDataInput INHRDataMeas INHRDataOutput INHRDataPgm INHRDataUUTComm	NHRDataMgr.NHRData	Automation Objects NHR Component
Fault Manager	NHRFaultMgr.dll	NHRFault	INHRFault	NHRFault.NHRFault	Automation Objects NHR Component
CAN Communications	NHRCanSvr.exe	NHRCanComm	INHRCanComm INHRUUTComm	NHRCanSvr.NHRCanComm	Automation Objects NHR Component NHR UUT Communication
Serial Communications	NHRSerialSvr.exe	NHRSerialComm	INHRSerialComm INHRUUTComm	NHRSerialSvr.NHRSerialComm	Automation Objects NHR Component
Output (Load) Container	NHROutputSvr.dll	NHROutputContainer	INHRContainer INHRDmm INHRMeas INHROutputContainer	NHROutputSvr.NHROutputContainer	Automation Objects NHR Component NHR Container
Input (Source) Container	NHRInputSvr.dll	NHRInputContainer	INHRContainer INHRDmm INHRInputContainer INHRMeas	NHRInputSvr.NHRInputContainer	Automation Objects NHR Component NHR Container
Digital Container	NHRDigitalSvr.dll	NHRDigitalContainer	INHRContainer INHRDigitalContainer	NHRDigitalSvr.NHRDigitalContainer	Automation Objects NHR Component NHR Container
Test Information	NHRTestInfoSvr.dll		INHRTestInfo	NHRTestInfoSvr.NHRTestInfo	Automation Objects

Helper Server	NHRHelperSvr.dll	NHRHelper	INHRHelper	NHRHelperSvr.NHRHelper	Automation Objects NHR Component
NHR AC/DC Source	NHRDev1Svr.dll	NHRDev1	INHRDev INHRDev1 INHRDmm INHRMeas INHRSource	NHRDev1Svr.NHRDev1	Automation Objects NHR Component NHR Device
NHR S6K AC Load	NHRDev2Svr.dll	NHRDev2	INHRDev INHRDev2 INHRDmm INHRLoad INHRMeas	NHRDev2Svr.NHRDev2	Automation Objects NHR Component NHR Device
NHR S6K DMS	NHRDev3Svr.dll	NHRDev3	INHRDev INHRDev3 INHRDev3Maint INHRDigital INHRDin INHRDmm INHRMeas INHRMux INHRSysControl INHRSysMux INHRTiming INHRTransient	NHRDev3Svr. NHRDev3	Automation Objects NHR Component NHR Device
NHR 525 OVPS	NHRDev4Svr.dll	NHRDev4	INHRDev INHRDev4 INHRSource	NHRDev4Svr. NHRDev4	Automation Objects NHR Component NHR Device
NHR 525 AC Source	NHRDev5Svr.dll	NHRDev5	INHRDev INHRDev5 INHRDmm INHRMeas INHRSource	NHRDev5Svr. NHRDev5	Automation Objects NHR Component NHR Device
NHR 525 DC Load	NHRDev6Svr.dll	NHRDev6	INHRDev INHRDev6 INHRDmm INHRLoad INHRMeas	NHRDev6Svr. NHRDev6	Automation Objects NHR Component NHR Device
NHR 5x5 Measurement	NHRDev8Svr.dll	NHRDev8	INHRDev INHRDev8 INHRDmm INHRMeas INHRMux INHRSysMux INHRTransient	NHRDev8Svr. NHRDev8	Automation Objects NHR Component NHR Device

HP 34401A DMM	NHRDev9Svr.dll	NHRDev9	INHRDev INHRDev9 INHRDmm INHRMeas	NHRDev9Svr. NHRDev9	Automation Objects NHR Component NHR Device
NHR S6K DC Source	NHRDev10Svr.dll	NHRDev10	INHRDev INHRDev10 INHRDmm INHRMeas INHRSource	NHRDev10Svr. NHRDev10	Automation Objects NHR Component NHR Device
NHR S6K DC Load	NHRDev11Svr.dll	NHRDev11	INHRDev INHRDev11 INHRDmm INHRLoad INHRMeas	NHRDev11Svr. NHRDev11	Automation Objects NHR Component NHR Device
CIC L Series AC Source	NHRDev12Svr.dll	NHRDev12	INHRDev INHRDev12 INHRDmm INHRMeas INHRSource	NHRDev12Svr. NHRDev12	Automation Objects NHR Component NHR Device
Yokogawa WT110/WT130 Power Meter	NHRDev13Svr.dll	NHRDev13	INHRDev INHRDev13 INHRDmm INHRMeas	NHRDev13Svr. NHRDev13	Automation Objects NHR Component NHR Device
HP 34970A Data Acquisition / Switch Unit	NHRDev14Svr.dll	NHRDev14	INHRDev INHRDev14 INHRDmm INHRMeas INHRMux	NHRDev14Svr. NHRDev14	Automation Objects NHR Component NHR Device
EMI ESS DC Source	NHRDev15Svr.dll	NHRDev15	INHRDev INHRDev15 INHRDmm INHRMeas INHRSource	NHRDev15Svr. NHRDev15	Automation Objects NHR Component NHR Device
NHR PWM 1/3PH	NHRDev16Svr.dll	NHRDev16	INHRAnalog INHRDev INHRDev16	NHRDev16Svr. NHRDev16	Automation Objects NHR Component NHR Device

5. API REFERENCE

5.1 Members of NHRINTERFACESLib.INHRAnalog

- NHR General Analog Interface
- Property PrimaryValue(lLogID As Long) As String - Set or get the primary value of the device in string form.

5.2 Members of NHRINTERFACESLib.INHRContainer

- NHR General Container Interface
- Property ActiveFixture As FIXTURE_ID - Set or get the active fixture of the container when in multi-fixture mode.
- Function AddDevice(sLogicalName As String) As Long - Adds the specified device to the container.
- Function ApplyAndTriggerBlobProp(pbTriggered As Long) As Long - Apply the property blob to hardware. Returns TRUE if successful. If pbTriggered returns TRUE, hardware was also triggered.
- Function ApplyBlobCfg() As Long - Applies the configuration blob to hardware.
- Function ApplyBlobProp() As Long - Applies the property blob to hardware.
- Property BlobMgr As Unknown - Set to the Blob Manager object instance (write only).
- Property CfgMgr As Unknown - Set to the Configuration Manager object instance (write only).
- Function CopyBlobProp(hBlob As Long) As Long - Copies the property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.
- Function DeleteBlobCfg(hBlob As Long) As Long - Delete the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function DeleteBlobProp(hBlob As Long) As Long - Delete the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property Describe As String – User-defined description.
- Property DeviceCount As Long - Number of devices under control of this container (read only).
- Function DumpCfg([bPrinterFormat As Long = -1]) As String - Human readable dump of the configuration. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Function DumpProp([bPrinterFormat As Long = -1]) As String - Human readable dump of the properties. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.

- Property FaultMgr As Unknown - Set to the Fault Manager object instance (write only).
- Function GetDevice(lIndex As Long, plLogID As Long) As Unknown - Returns a pointer to IUnknown of the specified device in the container.
- Function LoadBlobCfg(hBlob As Long) As Long - Load the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function LoadBlobProp(hBlob As Long) As Long - Load the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property MultiFixture As Long - Set or get whether the container is in multi-fixture mode.
- Property PrimaryValue As String - Set or get the primary value of the container in string form.
- Function ReconcileBlobMeas(hBlob As Long) As Long - Adjust the measurement property data referred to by hBlob to match the container configuration. Returns TRUE if successful.
- Function ReconcileBlobProp(hBlob As Long) As Long - Adjust the property data referred to by hBlob to match the container configuration. Returns TRUE if successful.
- Sub RefreshBlobProp() - Set blob according to current hardware state.
- Function RemoveAllDevices() As Long - Removes all devices from the container.
- Function RemoveDevice(sLogicalName As String) As Long - Removes the specified device from the container.
- Function SaveBlobCfg(hBlob As Long) As Long - Saves the configuration data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SaveBlobProp(hBlob As Long) As Long - Saves the property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SetToDefault() As Long - Sets the configuration of the container to its default state.
- Sub SetToOffState() - Sets the devices in the container to the off state.
- Function UICfg() As Long - Display the configuration user interface. Returns TRUE if OK pressed.
- Function UIProp() As Long - Display the property user interface. Returns TRUE if OK pressed.
- Function UIVIP(pHWnd As Long) As Long - Display the basic VIP user interface.
- Function UIVIPAdv() As Long - Displays the advanced VIP user interface.
- Function UIVIPDebug() As Long - Displays the debug VIP user interface.
- Property WizardMode As Long - TRUE for wizard mode, FALSE for tabbed dialog mode (write only).

5.3 Members of NHRINTERFACESLib.INHRDataAnalog

- INHRDataAnalog Interface
- Function CreateAnalogDevice(lIndex As Long) As Long - Creates a new device. Returns the id of the new device.
- Function CreateAnalogVector(lIndex As Long) As Long - Creates a new analog vector. Returns the id of the new vector.
- Sub DestroyAnalogDevice(lIdDevice As Long) - Destroys the device identified by id.
- Sub DestroyAnalogVector(lIdVector As Long) - Destroys the analog vector identified by id.
- Function DuplicateAnalogDevice(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function DuplicateAnalogVector(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function GetAnalogDeviceId(lIndex As Long) As Long - Returns the id for the device at lIndex (0 based). Returns 0 on error.
- Function GetAnalogDeviceLogName(lIdDevice As Long) As String - Gets the LogName associated with the device id.
- Function GetAnalogVectorId(lIndex As Long) As Long - Returns the id for the analog vector at lIndex (0 based). Returns 0 on error.
- Function GetAnalogVectorLabel(lIdVector As Long) As String - Gets the label for the vector id.
- Function GetAnalogVectorProp(lIdVector As Long, lIdDevice As Long) As Long - Gets the Analog container's property blob handle for the vector id.
- Function GetNumAnalogVectors() As Long - Number of analog control vectors.
- Property NumAnalogDevices As Long - Number of devices (read only).
- Sub ReorderAnalogDevice(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub ReorderAnalogVector(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetAnalogDeviceLogName(lIdDevice As Long, sLogName As String) - Sets the LogName associated with the device id.
- Sub SetAnalogVectorLabel(lIdVector As Long, sLabel As String) - Sets the label for the vector id.
- Sub SetAnalogVectorProp(lIdVector As Long, lIdDevice As Long, lHBlob As Long) - Sets the Analog container's property blob handle for the vector id.

5.4 Members of NHRINTERFACESLib.INHRDataDatalog

- INHRDataDatalog Interface
- Function CreateDatalogDrv(lIndex As Long, sProgId As String) As Long - Creates a new driver from the ProgId. Returns the id of the new driver.
- Sub DestroyDatalogDrv(lIdDriver As Long) - Destroys the driver identified by id.
- Sub DestroyDatalogDrv(lIdDriver As Long) - Destroys the driver identified by id.
- Function GetDatalogDrvCfg(lIdDriver As Long) As Long - Gets the configuration blob handle associated with the driver id.
- Function GetDatalogDrvId(lIndex As Long) As Long - Returns the id for the driver at lIndex (0 based). Returns 0 on error.
- Function GetDatalogDrvObj(lIdDriver As Long) As Unknown - Gets the IUnknown interface for the driver associated with the driver id.
- Property NumDatalogDrvs As Long - Number of drivers (read only).
- Sub ReorderDatalogDrv(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetDatalogDrvCfg(lIdDriver As Long, lHBlob As Long) - Sets the configuration blob handle associated with the driver id.

5.5 Members of NHRINTERFACESLib.INHRDataDigital

- INHRDataDigital Interface
- Function CreateDigitalContainer(lIndex As Long) As Long - Creates a new container. Returns the id of the new container.
- Function CreateDigitalVector(lIndex As Long) As Long - Creates a new vector. Returns the id of the new vector.
- Sub DestroyDigitalContainer(lIdContainer As Long) - Destroys the container identified by id.
- Sub DestroyDigitalVector(lIdVector As Long) - Destroys the vector identified by id.
- Function DuplicateDigitalContainer(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function DuplicateDigitalVector(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function GetDigitalContainerCfg(lIdContainer As Long) As Long - Gets the configuration blob handle associated with the container id.
- Function GetDigitalContainerId(lIndex As Long) As Long - Returns the id for the container at lIndex (0 based). Returns 0 on error.
- Function GetDigitalContainerObj(lIdContainer As Long) As Unknown - Gets the IUnknown interface for the container associated with the container id.

- Function GetDigitalContainerProp(lIdVector As Long, lIdContainer As Long) As Long - Gets the container's property blob handle for the vector id and the container id.
- Function GetDigitalVectorId(lIndex As Long) As Long - Returns the id for the vector lIndex (0 based). Returns 0 on error.
- Function GetDigitalVectorLabel(lIdVector As Long) As String - Gets the label for the vector id.
- Property NumDigitalContainers As Long - Number of containers (read only).
- Property NumDigitalVectors As Long - Number of vectors (read only).
- Sub ReorderDigitalContainer(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub ReorderDigitalVector(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetDigitalContainerCfg(lIdContainer As Long, lHBlob As Long) - Sets the configuration blob handle associated with the container id.
- Sub SetDigitalContainerProp(lIdVector As Long, lIdContainer As Long, lHBlob As Long) - Sets the container's property blob handle for the vector id and the container id.
- Sub SetDigitalVectorLabel(lIdVector As Long, sLabel As String) - Sets the label for the vector id.

5.6 Members of NHRINTERFACESLib.INHRDataInput

- INHRDataInput Interface
- Function CreateInContainer(lIndex As Long) As Long - Creates a new container. Returns the id of the new container.
- Function CreateInVector(lIndex As Long) As Long - Creates a new vector. Returns the id of the new vector.
- Sub DestroyInContainer(lIdContainer As Long) - Destroys the container identified by id.
- Sub DestroyInVector(lIdVector As Long) - Destroys the vector identified by id.
- Function DuplicateInContainer(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function DuplicateInVector(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function GetInAnalogCol() As Long - Gets the column associated with the Analog vectors.
- Function GetInAnalogVectorId(lIdVector As Long) As Long - Gets the Analog vector id for the vector id.
- Function GetInContainerCfg(lIdContainer As Long) As Long - Gets the configuration blob handle associated with the container id.

- Function GetInContainerId(lIndex As Long) As Long - Returns the id for the container at lIndex (0 based). Returns 0 on error.
- Function GetInContainerObj(lIdContainer As Long) As Unknown - Gets the IUnknown interface for the container associated with the container id.
- Function GetInContainerProp(lIdVector As Long, lIdContainer As Long) As Long - Gets the container's property blob handle for the vector id and the container id.
- Function GetInDigitalCol() As Long - Gets the column associated with the digital vectors.
- Function GetInDigitalVectorId(lIdVector As Long) As Long - Gets the digital vector id for the vector id.
- Function GetInOffVector(lIdVector As Long) As Long - TRUE to define this vector as an OFF vector.
- Function GetInVectorId(lIndex As Long) As Long - Returns the id for the vector at lIndex (0 based). Returns 0 on error.
- Function GetInVectorLabel(lIdVector As Long) As String - Gets the label for the vector id.
- Property InCRDelay As Double - If enabled, the time that the loads are in CR mode after going from an 'OFF' to an 'ON' input vector.
- Property InEnableAutoCRAtTurnOn As Long - If true, the loads will start in CR mode after going from an 'OFF' to an 'ON' input vector.
- Property InMinimumOffTime As Double - Property for the minimum time that should pass between an 'OFF' and an 'ON' input vector.
- Property InReverseTurnOffSequence As Long - Property that indicates whether the start-up sequence is to be reversed when shutting down.
- Property NumInContainers As Long - The number of containers (read only).
- Property NumInVectors As Long - Number of vectors (read only).
- Sub ReorderInContainer(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub ReorderInVector(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetInAnalogCol(lCol As Long) - Sets the column associated with the Analog vectors.
- Sub SetInAnalogVectorId(lIdVector As Long, lIdAnalogVector As Long) - Sets the Analog vector id for the vector id.
- Sub SetInContainerCfg(lIdContainer As Long, lHBlob As Long) - Sets the configuration blob handle associated with the container id.
- Sub SetInContainerProp(lIdVector As Long, lIdContainer As Long, lHBlob As Long) - Sets the container's property blob handle for the vector id and the container id.
- Sub SetInDigitalCol(lCol As Long) - Sets the column associated with the digital vectors.

- Sub SetInDigitalVectorId(lIdVector As Long, lIdDigitalVector As Long) - Sets the digital vector id for the vector id.
- Sub SetInOffVector(lIdVector As Long, bOffVector As Long) - TRUE to define this vector as an 'OFF' vector.
- Sub SetInVectorLabel(lIdVector As Long, sLabel As String) - Sets the label for the vector id.

5.7 Members of NHRINTERFACESLib.INHRDataMeas

- INHRDataMeas Interface
- Function CreateMeasVector(lIndex As Long) As Long - Creates a new vector. Returns the id of the new vector.
- Sub DestroyMeasVector(lIdVector As Long) - Destroys the vector identified by id.
- Function DuplicateMeasVector(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function GetMeasDevName(lIdVector As Long) As String - Gets the logical device name of the measurement device for the vector id.
- Function GetMeasDevProp(lIdVector As Long) As Long - Gets the property blob handle for the vector id.
- Function GetMeasVectorId(lIndex As Long) As Long - Returns the id for the vector at lIndex (0 based). Returns 0 on error.
- Function GetMeasVectorLabel(lIdVector As Long) As String - Gets the label for the vector id.
- Property NumMeasVectors As Long - The number of vectors(read only).
- Sub ReorderMeasVector(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetMeasDevName(lIdVector As Long, sLogDevName As String) - Sets the logical device name of the measurement device for the vector id.
- Sub SetMeasDevProp(lIdVector As Long, lHBlob As Long) - Sets the property blob handle for the vector id.
- Sub SetMeasVectorLabel(lIdVector As Long, sLabel As String) - Sets the label for the vector id.

5.8 Members of NHRINTERFACESLib.INHRDataOutput

- INHRDataOutput Interface
- Function CreateOutContainer(lIndex As Long) As Long - Creates a new container. Returns the id of the new container.
- Function CreateOutVector(lIndex As Long) As Long - Creates a new vector. Returns the id of the new vector.
- Sub DestroyOutContainer(lIdContainer As Long) - Destroys the container identified by id.
- Sub DestroyOutVector(lIdVector As Long) - Destroys the vector identified by id.
- Function DuplicateOutContainer(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function DuplicateOutVector(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function GetOutAnalogCol() As Long - Gets the column associated with the Analog vectors.
- Function GetOutAnalogVectorId(lIdVector As Long) As Long - Gets the Analog vector id for the vector id.
- Function GetOutContainerCfg(lIdContainer As Long) As Long - Gets the configuration blob handle associated with the container id.
- Function GetOutContainerId(lIndex As Long) As Long - Returns the id for the container at lIndex (0 based). Returns 0 on error.
- Function GetOutContainerObj(lIdContainer As Long) As Unknown - Gets the IUnknown interface for the container associated with the container id.
- Function GetOutContainerProp(lIdVector As Long, lIdContainer As Long) As Long - Gets the container's property blob handle for the vector id and the container id.
- Function GetOutDigitalCol() As Long - Gets the column associated with the digital vectors.
- Function GetOutDigitalVectorId(lIdVector As Long) As Long - Gets the digital vector id for the vector id.
- Function GetOutVectorId(lIndex As Long) As Long - Returns the id for the vector lIndex (0 based). Returns 0 on error.
- Function GetOutVectorLabel(lIdVector As Long) As String - Gets the label for the vector id.
- Property NumOutContainers As Long - The number of containers (read only).
- Property NumOutVectors As Long - The number of vectors (read only).
- Sub ReorderOutContainer(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub ReorderOutVector(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetOutAnalogCol(lCol As Long) - Sets the column associated with the Analog vectors.

- Sub SetOutAnalogVectorId(lIdVector As Long, lIdAnalogVector As Long) - Sets the Analog vector id for the vector id.
- Sub SetOutContainerCfg(lIdContainer As Long, lHBlob As Long) - Sets the configuration blob handle associated with the container id.
- Sub SetOutContainerProp(lIdVector As Long, lIdContainer As Long, lHBlob As Long) - Sets the container's property blob handle for the vector id and the container id.
- Sub SetOutDigitalCol(lCol As Long) - Sets the column associated with the digital vectors.
- Sub SetOutDigitalVectorId(lIdVector As Long, lIdDigitalVector As Long) - Sets the digital vector id for the vector id.
- Sub SetOutVectorLabel(lIdVector As Long, sLabel As String) - Sets the label for the vector id.

5.9 Members of NHRINTERFACESLib.INHRDataPgm

- INHRDataPgm Interface
- Function CreatePgmTestStep(lIndex As Long, pgmtype As PROGRAM_TYPE) As Long - Creates a new test step. Returns the id of the new test step.
- Sub DestroyPgmTestStep(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) - Destroys the test step identified by id.
- Function DuplicatePgmTestStep(lFromIndex As Long, lToIndex As Long, pgmtype As PROGRAM_TYPE) As Long - Duplicates the lFromIndex test step into the lToIndex. Returns the id of the new test step.
- Function GetPgmAnalogVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As Long - Gets the id of the Analog vector for the test step id.
- Function GetPgmDigitalVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As Long - Gets the id of the Digital vector for the test step id.
- Function GetPgmExecuteState(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As EXECUTE_STATE - Gets the execute state for the test step id.
- Function GetPgmInputVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As Long - Gets the id of the input vector for the test step id.
- Function GetPgmOutputVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As Long - Gets the id of the output vector for the test step id.
- Function GetPgmSequenceProp(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As Long - Gets the sequencer property blob handle for the test step id.
- Function GetPgmTestRoutineProgId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As String - Gets the ProgId of the test routine for the test step id.
- Function GetPgmTestRoutineProp(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As Long - Gets the test routine property blob handle for the test step id.

- Function GetPgmTestStepId(pgmtype As PROGRAM_TYPE, lIndex As Long) As Long - Returns the id for the test step at lIndex (0 based). Returns 0 on error.
- Function GetPgmTestStepLabel(pgmtype As PROGRAM_TYPE, lIdTestStep As Long) As String - Gets the label for test step id.
- Function NumPgmTestSteps(pgmtype As PROGRAM_TYPE) As Long - Number of test steps.
- Sub ReorderPgmTestStep(lOldIndex As Long, lNewIndex As Long, pgmtype As PROGRAM_TYPE) - Changes the order from lOldIndex to lNewIndex.
- Sub SetPgmAnalogVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, lIdVector As Long) - Sets the id of the Analog vector for the test step id.
- Sub SetPgmDigitalVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, lIdVector As Long) - Sets the id of the Digital vector for the test step id.
- Sub SetPgmExecuteState(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, executestate As EXECUTE_STATE) - Sets the execute state for the test step id.
- Sub SetPgmInputVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, lIdVector As Long) - Sets the id of the input vector for the test step id.
- Sub SetPgmOutputVectorId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, lIdVector As Long) - Sets the id of the output vector for the test step id.
- Sub SetPgmSequenceProp(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, lHBlob As Long) - Sets the sequencer property blob handle for the test step id.
- Sub SetPgmTestRoutineProgId(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, sTestRoutineProgId As String) - Sets the ProgId of the test routine for the test step id.
- Sub SetPgmTestRoutineProp(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, lHBlob As Long) - Sets the test routine property blob handle for the test step id.
- Sub SetPgmTestStepLabel(pgmtype As PROGRAM_TYPE, lIdTestStep As Long, sLabel As String) - Sets the label for test step id.

5.10 Members of NHRINTERFACESLib.INHRDataUUTComm

- INHRDataUUTComm Interface
- Function CreateUUTCommObj(lIndex As Long, sProgId As String) As Long - Creates a new driver from the ProgId. Returns the id of the new driver.
- Sub DestroyUUTCommObj(lIdDriver As Long) - Destroys the driver identified by id.
- Function DuplicateUUTCommObj(lFromIndex As Long, lToIndex As Long) As Long - Copies the lFromIndex to the lToIndex. Returns the id.
- Function GetUUTCommObj(lIdDriver As Long) As Unknown - Gets the IUnknown interface for the driver associated with the driver id.
- Function GetUUTCommObjCfg(lIdDriver As Long) As Long - Gets the configuration blob handle associated with the driver id.

- Function GetUUTCommObjId(lIndex As Long) As Long - Returns the id for the driver at lIndex (0 based). Returns 0 on error.
- Property NumUUTCommObjs As Long - Number of drivers (read only).
- Sub ReorderUUTCommObj(lOldIndex As Long, lNewIndex As Long) - Changes the order from lOldIndex to lNewIndex.
- Sub SetUUTCommObjCfg(lIdDriver As Long, lHBlob As Long) - Sets the configuration blob handle associated with the driver id.

5.11 Members of NHRINTERFACESLib.INHRDev

- NHR General Device Interface
- Property Address As Long - Address of the device.
- Sub AddSlave(lpunkSlave As Unknown) - Slave device's object instance.
- Function ApplyAndTriggerBlobProp(lLogID As Long, pbTriggered As Long) As Long - Apply the properties blob to hardware. Returns TRUE if successful. If pbTriggered returns TRUE, hardware was also triggered.
- Sub ApplyBlobCfg(lLogID As Long) - Applies the configuration blob to hardware.
- Function ApplyBlobProp(lLogID As Long) As Long - Apply the properties blob to hardware and a trigger. Returns TRUE if successful.
- Function ApplyBlobPropAdv(lLogID As Long) As Long - Apply the advanced properties blob to hardware.
- Property CacheState As Long - 0 = cache disabled, 1 = cache enabled.
- Property CfgBlobMgr As Unknown - Set to the blob manager object instance used for config data (write only).
- Property CfgSvr As Unknown - Set to the configuration server object instance used for config data (write only).
- Function CopyBlobProp(hBlob As Long, lLogID As Long) As Long - Copies the property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.
- Function CopyBlobPropAdv(hBlob As Long, lLogID As Long) As Long - Copies the advanced property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.
- Function DeleteBlobCfg(hBlob As Long) As Long - Delete the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function DeleteBlobProp(hBlob As Long, lLogID As Long) As Long - Delete the property data referred to by hBlob from the blob manager. Returns TRUE if successful.

- Function DeleteBlobPropAdv(hBlob As Long, lLogID As Long) As Long - Delete the advanced property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property DescribeCfg As String - Description of the device configuration (read only).
- Property DescribeDevice As String - Terse description of the device (read only).
- Property DescribeInDetail As String - Detailed description of the device (read only).
- Property DeviceID As Long - Unique ID to distinguish this device from all other NHR device types (read only).
- Function DumpCfg([bPrinterFormat As Long = -1]) As String - Human readable dump of the configuration. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Function DumpProp([bPrinterFormat As Long = -1], lLogID As Long) As String - Human readable dump of the properties. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Function DumpPropAdv([bPrinterFormat As Long = -1], lLogID As Long) As String - Human readable dump of the advanced properties. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Property FaultMgr As Unknown - Set to the Fault object instance (write only).
- Function GetDescribe(lLogID As Long) As String – User-defined description.
- Function GetInUse(lLogID As Long) As Long - TRUE if device has been used in a container.
- Function GetLogicalDevSubType(lLogID As Long) As String - The sub-type of the logical device. Returns "GPR", "Din", "Dout", "TA", etc (read only)..
- Function GetLogicalDevType(lLogID As Long) As String - The type of the logical device. Returns "Control", "Measurement", "Load", "Source", etc (read only)..
- Function HasPropAdv(lLogID As Long) As Long - TRUE if the device has advanced properties (read only).
- Function HasUIFirmwareUpdate(lLogID As Long) As Long - TRUE if the device has firmware update capability (read only).
- Property HostCfgBlob As Long - The configuration blob handle for my host.
- Property IsConfigured As Long - TRUE if UICfg has been called and exited with an OK (read only).
- Property IsParallelable As Long - TRUE if this device can be paralleled with identical devices (read only).
- Property LibBlobMgr As Unknown - Set to the blob manager object instance used for program data (stored in program library) (write only).
- Function LoadBlobCfg(hBlob As Long) As Long - Load the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.

- Function LoadBlobProp(hBlob As Long, lLogID As Long) As Long - Load the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function LoadBlobPropAdv(hBlob As Long, lLogID As Long) As Long - Load the advanced property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property NumLogicalDevs As Long - Number of logical devices that this device supports (read only).
- Sub RefreshBlobCfg(lLogID As Long) - Sets the configuration blob according to current hardware state.
- Sub RefreshBlobProp(lLogID As Long) - Set blob according to current hardware state.
- Sub RefreshBlobPropAdv(lLogID As Long) - Set advanced blob according to current hardware state.
- Sub ReleaseCfg(lLogID As Long) - Instructs the device to free any resources before being removed from the configuration list.
- Sub RemoveSlave(lpunkSlave As Unknown) - Slave device's object instance. NULL to remove all slaves.
- Sub Reset([lLogID As Long = -1]) - Reset the device. Reset state is hardware dependent.
- Sub ResetCache() - Resets cache. Cache contents are re-established as commands are executed.
- Function SaveBlobCfg(hBlob As Long) As Long - Saves the configuration data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SaveBlobProp(hBlob As Long, lLogID As Long) As Long - Saves the property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SaveBlobPropAdv(hBlob As Long, lLogID As Long) As Long - Saves the advanced property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Sub SetDescribe(lLogID As Long, newVal As String) - A user defined description.
- Sub SetInUse(lLogID As Long, bInUse As Long) - TRUE if device has been used in a container.
- Sub SetToOffState([lLogID As Long = -1]) - Sets the device to the off state. The off state is hardware dependent.
- Property SlaveOf As Unknown - The master device's object instance. NULL if not paralleled.
- Function UICfg(lDeviceId As Long) As Long - Display the configuration user interface. Returns TRUE if OK pressed.

- Function UIFirmwareUpdate(lLogID As Long) As Long - Display the firmware update user interface.
- Function UIProp(lLogID As Long) As Long - Display the properties user interface.
- Function UIPropAdv(lLogID As Long) As Long - Display the advanced properties user interface.
- Function UIVIP(pHWnd As Long, lLogID As Long) As Long - Display the basic VIP user interface.
- Function UIVIPAdv(lLogID As Long) As Long - Displays the advanced VIP user interface.
- Function UIVIPDebug(lLogID As Long) As Long - Displays the debug VIP user interface.
- Function ValidateCfg() As Long - Perform a self-validation of configuration data. Returns TRUE if OK.

5.12 Members of NHRINTERFACESLib.INHRDigital

- NHR General Digital Interface
- Sub Apply(lLogID As Long, [bWaitForComplete As Long]) - Applies the preset states of the bits.
- Sub ApplyAndTrigger(lLogID As Long, pbTriggered As Long, [bWaitForComplete As Long]) - Apply the preset states of the bits. If pbTriggered returns TRUE, hardware was also triggered.
- Sub GetBit(lLogID As Long, sBit As String, pDigitalState As DIGITAL_STATE) - Gets the state of a bit (read only).
- Function NumBits(lLogID As Long) As Long - Returns the number Digital bits that can be set (read only).
- Sub PresetBit(lLogID As Long, sBit As String, DigitalState As DIGITAL_STATE) - Presets a bit to a state (write only).
- Sub SetBit(lLogID As Long, sBit As String, DigitalState As DIGITAL_STATE, [bWaitForComplete As Long]) - Immediately sets a bit to a state (write only).

5.13 Members of NHRINTERFACESLib.INHRDin

- NHR General Din Interface
- Sub Abort(lLogID As Long) - Abort the measurement and end the measurement cycle without retrieving data.
- Function Fetch(lLogID As Long, sBit As String, pDigitalState As DIGITAL_STATE) As Long - Retrieve the data and end the measurement cycle.
- Sub Meas(lLogID As Long, [dTimeout As Double], [bSyncAll As Long = 1]) - Arm, and wait for acquisition event, if applicable.
- Function Setup(lLogID As Long, sBit As String, [dThreshold As Double = 1000000000], [hBlobPropAdv As Long]) As Long - Setup the specified Din for a measurement cycle.

5.14 Members of NHRINTERFACESLib.INHRDmm

- NHR DMM Interface
- Sub Abort(lLogID As Long) - Abort the measurement and end the measurement cycle without retrieving data.
- Sub ConnectSys(lLogID As Long, lChannel As Long) - Connects a system Mux physical input channel to the Dmm.
- Sub Disconnect(lLogID As Long) - Disconnects any input to the Dmm.
- Function Fetch(lLogID As Long, pdValue As Double, [lNumMeasurements As Long = 1]) As Long - Retrieve the data and end the measurement cycle.
- Function IsFunctionValid(lLogID As Long, Function As DMM_FUNCTION) As Long - Validates the provided measurement function. Returns TRUE if supported; FALSE if not supported.
- Function IsFunctionValidEx(lLogID As Long, Function As DMM_FUNCTION) As Long - Validates the provided measurement function. Allows access to functions requiring special setups and/or internal connections. Returns TRUE if supported; FALSE if not supported.
- Sub Meas(lLogID As Long, [dTimeout As Double], [bSyncAll As Long = 1]) - Arm, and wait for acquisition event, if applicable.
- Function Setup(lLogID As Long, sChannel As String, pFunction As DMM_FUNCTION, pdRange As Double, pdResolution As Double, phBlobPropAdv As Long, [lNumMeasurements As Long = 1]) As Long - Setup the specified Dmm function.
- Function SetupBlobs(lLogID As Long, sChannel As String, phBlob As Long, pdRange As Double, [lNumMeasurements As Long = 1]) As Long - Setup the specified Dmm function using Blobs.

5.15 Members of NHRINTERFACESLib.INHRHost

- NHR General Host Interface
- Function Autofill() As Long - Establish all detectable devices on the host.
- Property BlobMgr As Unknown - Set to the blob manager object instance (write only).
- Sub ClosePort() - Closes the communication port.
- Function DeleteBlobCfg(hBlob As Long) As Long - Delete the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property DescribeCfg As String - A description of the host configuration (read only).
- Property DescribeInDetail As String - A detailed description of the host (read only).
- Property DevID As Long - The ID of the port Dev (read only).
- Function DumpCfg([bPrinterFormat As Long = -1]) As String - Human readable dump of the configuration. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Property FaultMgr As Unknown - Set to the Fault object instance (write only).

- Property IsConfigured As Long - TRUE if UICfg has been called and exited with an OK (read only).
- Property IsPortOpen As Long - TRUE if OpenPort has been called successfully (read only).
- Property IsValidDevice(lDeviceId As Long) As Long - TRUE if DeviceId is supported on this host (read only).
- Function LoadBlobCfg(hBlob As Long) As Long - Load the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Sub Reset() - Reset the host and flush all buffers.
- Function SaveBlobCfg(hBlob As Long) As Long - Saves the configuration data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Sub SynchronizeCommands() - Wait for all communication to all devices to be completed before continuing.
- Property SystemTriggerMaster As Long - This host is the system trigger source. (Read-Write)
- Sub Trigger() - Sends the system trigger.
- Function UICfg() As Long - Display the configuration user interface. Returns TRUE if OK pressed.
- Function UIVIP() As Long - Display the basic VIP user interface.
- Function UIVIPAdv() As Long - Displays the advanced VIP user interface.
- Function UIVIPDebug() As Long - Displays the debug VIP user interface.

5.16 Members of NHRINTERFACESLib.INHRList

- NHR ListView Interface
- Function BeginActiveSession() As Long - Make this call to gain access to the tempfile.
- Property Cancel As Long - Property Cancel.
- Function CreateLibrary(SCS As ShrikeComponentSync, obDataMngr As Object, sPath As String, sProgName As String) As Long - Create a new library and program and load it into datamanger
- Sub EndActiveSession() - Make this call to release the tempfile so others can gain access to it.
- Property FullPath As String - Property FullPath.
- Function IDKey() As Long - Method IDKey.
- Property LibrarySelected As String - Property LibrarySelected.
- Function LoadLibrary(SCS As ShrikeComponentSync, obDataMngr As Object, sPath As String) As Long - Load a library into data manager.

- Function LoadTempLibrary(SCS As ShrikeComponentSync, obDataMngr As Object) As Long - Sync data manager by reloading it with temporary file.
- Property LogonReadOnly As Long - Property LogOnReadOnly.
- Property OK As Long - Property OK.
- Property PathSelected As String - Property PathSelected.
- Property ProgramSelected As String - Property ProgramSelected.
- Function SaveLibrary(obDataMngr As Object, sPath As String) As Long - Commit temporary changes to the source library.
- Function SaveTempLibrary(SCS As ShrikeComponentSync, obDataMngr As Object, [DS As DirtySection = DS_MainTest]) As Long - Save changes to temporary file so *em*Power components will sync up to the changes.
- Function StartViewer(SCS As ShrikeComponentSync, obDataMngr As Object, sLib As String, CallerhWnd As Long, OpenForTheFirstTime As Long) As Long - Start the program browser: Takes a data manager pointer nulls are not ok here.
- Property UserLevel As String - Property UserLevel.
- Property UserName As String - Property UserName.

5.17 Members of NHRINTERFACESLib.INHRLoad

- NHR General Load Interface
- Function GetCurrentMax(lLogID As Long, [bGetMaxAmps As Long]) As Double - Maximum programmable current, in Amps (read only).
- Function GetCurrentMin(lLogID As Long) As Double - Minimum programmable current, in Amps (read only).
- Function GetCurrentRange(lLogID As Long) As Double - Get current range, in Amps.
- Function GetMode(lLogID As Long) As LOAD_MODES - Get operating mode, in numeric form.
- Function GetModeStr(lLogID As Long, [bVerbose As Long]) As String - Get operating mode, in string form.
- Function GetPeakWattageMax(lLogID As Long, [bGetMaxWatts As Long]) As Double - Maximum noncontinuous programmable power, in Watts (read only).
- Function GetValue(lLogID As Long) As Double - Get set point value (NOT a measurement). Units are dependent on the operating mode.
- Function GetValueStr(lLogID As Long, [bVerbose As Long]) As String - Get the programmed value, in string form. Units are dependent on the operating mode.
- Function GetVoltageMax(lLogID As Long, [bGetMaxVolts As Long]) As Double - Maximum programmable voltage, in Volts (read only).

- Function GetVoltageMin(lLogID As Long) As Double - Minimum programmable voltage, in Volts (read only).
- Function GetVoltageNom(lLogID As Long) As Double - Get nominal voltage, in Volts (NOT a measurement).
- Function GetVoltageRange(lLogID As Long) As Double - Get voltage range, in Volts.
- Function GetWattageMax(lLogID As Long, [bGetMaxWatts As Long]) As Double - Maximum continuous programmable power, in Watts (read only).
- Function GetWattageMin(lLogID As Long) As Double - Minimum programmable power, in Watts (read only).
- Function IsModeValid(lLogID As Long, lMode As LOAD_MODES) As Long - Validates the provided load mode. Returns TRUE if supported; FALSE if not supported.
- Function SetCurrentRange(lLogID As Long, dCurrent As Double, [bWaitForComplete As Long]) As Long - Set current range, in Amps.
- Function SetMode(lLogID As Long, lMode As LOAD_MODES, [bWaitForComplete As Long]) As Long - Set operating mode, in numeric form.
- Function SetModeStr(lLogID As Long, bstrMode As String, [bWaitForComplete As Long]) As Long - Set operating mode, in string form.
- Function SetValue(lLogID As Long, dValue As Double, [bWaitForComplete As Long]) As Long - Set value. Units are dependent on the operating mode.
- Function SetValueStr(lLogID As Long, bstrValue As String, [bWaitForComplete As Long]) As Long - Set the programmed value, in string form. Units are dependent on the operating mode.
- Function SetVoltageNom(lLogID As Long, dVolts As Double, [bWaitForComplete As Long]) As Long - Set nominal voltage, in Volts.
- Function SetVoltageRange(lLogID As Long, dVoltage As Double, [bWaitForComplete As Long]) As Long - Set voltage range, in Volts.

5.18 Members of NHRINTERFACESLib.INHRMeas

- NHR General Measurement Interface
- Function ApplyBlobMeas(lLogID As Long) As Long - Apply the measurement properties blob to hardware. Returns TRUE if successful.
- Function ApplyBlobMeasAdv(lLogID As Long) As Long - Apply the advanced measurement properties blob to hardware.
- Function CopyBlobMeas(hBlob As Long, lLogID As Long) As Long - Copies the measurement property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.

- Function CopyBlobMeasAdv(hBlob As Long, lLogID As Long) As Long - Copies the advanced measurement property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.
- Function DeleteBlobMeas(hBlob As Long, lLogID As Long) As Long - Deletes the measurement property data in the blob manager at the specified hBlob. Returns the TRUE if successful.
- Function DeleteBlobMeasAdv(hBlob As Long, lLogID As Long) As Long - Delete the advanced measurement property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function DumpMeas(lLogID As Long, [bPrinterFormat As Long = -1]) As String - Human readable dump of the measurement properties.
- Function DumpMeasAdv(lLogID As Long, [bPrinterFormat As Long = 1]) As String - Human readable dump of the advanced measurement properties. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Function GetDescribe(lLogID As Long) As String - Human readable description of the measurement.
- Function GetUnits(lLogID As Long) As String - Returns the units for the active measurement.
- Function HasMeasAdv([lLogID As Long]) As Long - TRUE if the device has advanced measurement properties (read only).
- Function LoadBlobMeas(hBlob As Long, lLogID As Long) As Long - Load the measurement property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function LoadBlobMeasAdv(hBlob As Long, lLogID As Long) As Long - Load the advanced measurement property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Sub RefreshBlobMeas(lLogID As Long) - Set blob according to current hardware state.
- Sub RefreshBlobMeasAdv(lLogID As Long) - Set advanced measurement blob according to current hardware state.
- Function SaveBlobMeas(hBlob As Long, lLogID As Long) As Long - Saves the measurement property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SaveBlobMeasAdv(hBlob As Long, lLogID As Long) As Long - Saves the advanced measurement property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function UIMeas(lLogID As Long) As Long - Display the measurement property user interface.

- Function UIMeasAdv(lLogID As Long, lFunction As Long) As Long - Display the advanced measurement properties user interface.

5.19 Members of NHRINTERFACESLib.INHRMux

- NHR User Mux Interface
- Function Connect(lLogID As Long, sChannel As String, lOutput As Long) As Long - Connects a logical input channel to the physical output.
- Sub Disconnect(lLogID As Long, lOutput As Long) - Disconnects any input from the physical output. (<0 for all).
- Function IsSupported(lLogID As Long, sChannel As String, lOutput As Long) As Long - Returns the connection depth (0 is not supported).
- Function Lock(lLogID As Long, sChannel As String, lOutput As Long, lMuxOwnership As Long, [lWaitMS As Long = -1]) As Long - Reserves exclusive use of a physical input channel path to a physical output.
- Function Unlock(lLogID As Long, sChannel As String, lOutput As Long) As Long - Releases exclusive use of a physical input channel path to a physical output.

5.20 Members of NHRINTERFACESLib.INHRSource

- NHR General Source Interface
- Function GetCurrentLimit(lLogID As Long, [lPhase As Long]) As Double - Get set point current limit (NOT a measurement).
- Function GetCurrentLimitMax(lLogID As Long) As Double - Maximum programmable current limit, in Amps (read only).
- Function GetCurrentLimitMin(lLogID As Long) As Double - Minimum programmable current limit, in Amps (read only).
- Function GetCurrentRange(lLogID As Long) As Double - Get current range.
- Function GetFrequency(lLogID As Long) As Double - Programmed frequency in Hertz (NA if DC).
- Function GetFrequencyMax(lLogID As Long) As Double - Maximum programmable frequency, in Hertz (NA if DC) (read only).
- Function GetFrequencyMin(lLogID As Long) As Double - Minimum programmable frequency, in Hertz (NA if DC) (read only).
- Function GetIsFreqProgrammable(lLogID As Long) As Long - TRUE if frequency is programmable (FALSE if DC) (read only).
- Function GetIsPhaseControllable(lLogID As Long) As Long - TRUE if phases can be individually controlled (FALSE if DC) (read only).
- Function GetIsPhaseSupported(lLogID As Long, lControl As PHASE_CONTROL) As Long - TRUE if supplied phase control can be set (FALSE if DC) (read only).

- Function GetNumPhases(lLogID As Long) As Long - How many phases this source supports (1 if DC) (read only).
- Function GetPhaseControl(lLogID As Long) As PHASE_CONTROL - Get the active phase mode and control of the source.
- Function GetPresenceSignalPolarity(lLogID As Long) As PRESENCE_SIGNAL - Polarity of the source's presence signal (0 if not supported) (read only).
- Function GetVoltage(lLogID As Long, [lPhase As Long]) As Double - Get set point voltage (NOT a measurement).
- Function GetVoltageMax(lLogID As Long) As Double - Maximum programmable voltage, in Volts (read only).
- Function GetVoltageMin(lLogID As Long) As Double - Minimum programmable voltage, in Volts (read only).
- Function GetVoltageRange(lLogID As Long) As Double - Get voltage range.
- Function GetWattageMax(lLogID As Long, [bGetMaxWatts As Long]) As Double - Maximum programmable power, in Watts (read only).
- Function GetWattageMin(lLogID As Long) As Double - Minimum programmable power, in Watts (read only).
- Function SetCurrentLimit(lLogID As Long, dCurrent As Double, [lPhase As Long], [bWaitForComplete As Long]) As Long - Set current limit.
- Function SetCurrentRange(lLogID As Long, dCurrent As Double, [bWaitForComplete As Long]) As Long - Set current range.
- Function SetFrequency(lLogID As Long, dFrequency As Double, [bWaitForComplete As Long]) As Long - Programmed frequency in Hertz (NA if DC).
- Function SetPhaseControl(lLogID As Long, lControl As PHASE_CONTROL, [bWaitForComplete As Long]) As Long - Set the active phase mode and control of the source.
- Function SetVoltage(lLogID As Long, dVoltage As Double, [lPhase As Long], WaitForComplete As Long]) As Long - Set voltage.
- Function SetVoltageRange(lLogID As Long, dVoltage As Double, [bWaitForComplete As Long]) As Long - Set voltage range.

5.21 Members of NHRINTERFACESLib.INHRSysControl

- NHR System Control Interface
- Function GetBusyLight() As DIGITAL_STATE - Gets the light state.
- Function GetFailLight() As DIGITAL_STATE - Gets the light state.
- Function GetPassLight() As DIGITAL_STATE - Gets the light state.
- Function GetReadyLight() As DIGITAL_STATE - Gets the light state.

- Function GetStartButton() As DIGITAL_STATE - Get then clear the start button state.
- Function GetStartButtonAcknowledge() As DIGITAL_STATE - Gets the light state.
- Function GetStartLight() As DIGITAL_STATE - Gets the light state.
- Function GetStopButton() As DIGITAL_STATE - Get then clear the stop button state.
- Function GetStopButtonAcknowledge() As DIGITAL_STATE - Gets the light state.
- Function GetStopLight() As DIGITAL_STATE - Gets the light state.
- Function SetBusyLight(DigitalState As DIGITAL_STATE) As Long - Immediately sets the light to state specified.
- Function SetFailLight(DigitalState As DIGITAL_STATE) As Long - Immediately sets the light to state specified.
- Function SetPassLight(DigitalState As DIGITAL_STATE) As Long - Immediately sets the light to state specified.
- Function SetReadyLight(DigitalState As DIGITAL_STATE) As Long - Immediately sets the light to state specified.
- Function SetStartButtonAcknowledge(DigitalState As DIGITAL_STATE) As Long - Sets the button's acknowledge light to state specified.
- Function SetStartLight(DigitalState As DIGITAL_STATE) As Long - Immediately sets the light to state specified.
- Function SetStopButtonAcknowledge(DigitalState As DIGITAL_STATE) As Long - Sets the button's acknowledge light to state specified.
- Function SetStopLight(DigitalState As DIGITAL_STATE) As Long - Immediately sets the light to state specified.

5.22 Members of NHRINTERFACESLib.INHRSysMux

- NHR System Mux Interface
- Sub Connect(lLogID As Long, lChannel As Long, lOutput As Long) - Connects a physical input channel to the physical output.
- Sub Disconnect(lLogID As Long, lOutput As Long) - Disconnects any input from the physical output. (<0 for all).
- Function Lock(lLogID As Long, lChannel As Long, lOutput As Long, lMuxOwnership As Long, [lWaitMS As Long = -1]) As Long - Reserves exclusive use of a physical input channel path to a physical output.
- Function Unlock(lLogID As Long, lChannel As Long, lOutput As Long) As Long - Releases exclusive use of a physical input channel path to a physical output.

5.23 Members of NHRINTERFACESLib.INHRTestInfo

- NHR Test Information Interface
- Function DeleteBlobCfg(hBlob As Long) As Long - Delete the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function DumpCfg([bPrinterFormat As Long = -1]) As String - Human readable dump of the configuration. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Sub GetFieldInfo(lIndex As Long, psPrompt As String, psData As String) - 1Index is the 0 based index of the field to retrieve.
- Function GetNumFields() As Long - plNumFields will be the number of entry fields.
- Function GetTestInfo(psTestInfo As String) As Long - plStartFixture will be fixture index (0 or 1) or -1 if cancel.
- Sub Init(hwndParent As Long, pobCfgMgr As Object, pobDataMgr As Object, obFaultMgr As Object, obHelperSvr As Object, bOnline As Long)
- Function LoadBlobCfg(hBlob As Long) As Long
- Function LoadDialog(hWndCfg As Long, hWndEntry As Long) As Long - Call this to pre-load the testinfo prompt and retrieve the window handle.
- Function SaveBlobCfg(hBlob As Long) As Long - Sub SetZPos(lInsertAfter As Long)
- Function UICfg() As Long

5.24 Members of NHRINTERFACESLib.INHRTiming

- NHR Timing Interface
- Sub Abort(lLogID As Long) - Abort the measurement and end the measurement cycle without retrieving data.
- Sub Disconnect(lLogID As Long) - Disconnects any input to the Timing device.
- Function Fetch(lLogID As Long, pdValue As Double) As Long - Retrieve the data and end the measurement cycle.
- Function IsDeviceTypeValid(lLogID As Long, DeviceType As TIMING_DEVICE_TYPE) As Long - Validates the provided device type. Returns TRUE if supported; FALSE if not supported.
- Function IsDeviceTypeValidEx(lLogID As Long, DeviceType As TIMING_DEVICE_TYPE) As Long - Validates the provided device type. Allows access to device types requiring special setups and/or internal connections. Returns TRUE if supported; FALSE if not supported.
- Sub Meas(lLogID As Long, [dStartEventTimeout As Double], [bSyncAll As Long = 1]) - Start the measurement cycle and wait for acquisition event, if applicable.
- Function Setup(lLogID As Long, StartDeviceType As TIMING_DEVICE_TYPE, sStartDevice As String, bStartSlopeIsPositive As Long, dStartRange As Double, dStartLevel

As Double, StopDeviceType As TIMING_DEVICE_TYPE, sStopDevice As String, bStopSlopeIsPositive As Long, dStopRange As Double, dStopLevel As Double, dAperture As Double, dResolution As Double, hBlobAdvProp As Long) As Long - Setup the measurement cycle. Returns TRUE if the cycle was setup.

- Function SetupBlobs(lLogID As Long, hBlob As Long) As Long - Setup the specified Timing Measurement using a Blob.
- Function UsingSystemTrigger(lLogID As Long) As Long - TRUE if the current timing measurement is using a system trigger (read only).

5.25 Members of NHRINTERFACESLib.INHRTR

- NHR Test Routine Interface
- Sub CleanUp([sLogicalDevName As String]) - Remove data that is no longer valid.
- Function CopyBlobProp(hBlob As Long, [lLogID As Long]) As Long - Copies the property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.
- Function DeleteBlobProp(hBlob As Long, [lLogID As Long]) As Long - Delete the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property Describe As String - Description for the test routine (read only).
- Function DumpProp([bPrinterFormat As Long = -1], [lLogID As Long]) As String - Human readable dump of the properties. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Function Go(bDebug As Long, InitialVectorID As Long, [bAlwaysFail As Long]) As Long - Performs the test. Returns the number of errors.
- Sub Init(hwndParent As Long, pdispDatalogServer As Object, pdispDataManager As Object, pdispCfgManager As Object, pdispHelperSvr As Object, pdispFaultMgr As Object, pdispTestInfoSvr As Object) - Initializes the test routine.
- Function LoadBlobProp(hBlob As Long, [lLogID As Long]) As Long - Load the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function SaveBlobProp(hBlob As Long, [lLogID As Long]) As Long - Saves the property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Property SPC As Long - SPC in use flag (read only).
- Function UIProp() As Long - Display the properties user interface.

5.26 Members of NHRINTERFACESLib.INHRTransient

- NHR Transient Interface
- Sub Abort(lLogID As Long) - Abort the measurement and end the measurement cycle without retrieving data.
- Sub Disconnect(lLogID As Long) - Disconnects any input to the transient device.
- Function Fetch(lLogID As Long, pdValue As Double) As Long - Retrieve the data and end the measurement cycle.
- Function GetAperture(lLogID As Long) As Double - Gets the measurement window for transient operations, in seconds.
- Function IsFunctionValid(lLogID As Long, Function As TRANSIENT_FUNCTION) As Long - Validates the provided measurement function. Returns TRUE if supported; FALSE if not supported.
- Sub Meas(lLogID As Long, [dTimeout As Double], [bSyncAll As Long = 1]) - Arm, and wait for acquisition event, if applicable.
- Function Setup(lLogID As Long, sChannel As String, Function As TRANSIENT_FUNCTION, dRange As Double, dDelay As Double, dAperature As Double) As Long - Setup the specified Transient function.
- Function SetupBlobs(lLogID As Long, sChannel As String, Function As TRANSIENT_FUNCTION, dRange As Double, dDelay As Double, hBlob As Long) As Long - Setup the specified Transient function using Blob.

5.27 Members of NHRINTERFACESLib.INHRUUTComm

- NHR UUT Communications Interface
- Function ApplyBlobCfg() As Long - Applies the configuration blob to hardware.
- Function ApplyBlobProp() As Long - Applies the property blob to hardware.
- Function CopyBlobProp(hBlob As Long) As Long - Copies the property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed. Returns the hBlob of the new storage location if successful, 0 if failed.
- Function DeleteBlobCfg(hBlob As Long) As Long - Delete the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function DeleteBlobProp(hBlob As Long) As Long - Delete the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property Describe As String – User-defined description.
- Function DumpCfg([bPrinterFormat As Long = -1]) As String - Human readable dump of the configuration. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Function DumpProp([bPrinterFormat As Long = -1]) As String - A human readable dump of the properties. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.

- Function GetTestResults(pbstrResult As String, pbstrDescription As String) As RESULT_TYPE - Get the results from the test.
- Sub Init(pdispBlobMgr As Object, pdispFaultMgr As Object, bOffline As Long) - Initializes the UUT communication object.
- Function LoadBlobCfg(hBlob As Long) As Long - Load the configuration data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Function LoadBlobProp(hBlob As Long) As Long - Load the property data referred to by hBlob from the blob manager. Returns TRUE if successful.
- Property MaximumString As String - Set or get the expected maximum string value.
- Property MaximumValue As String - Set or get the expected maximum value.
- Property MaxReadLength As Long - Set or get the maximum number of characters in the response.
- Property MinimumString As String - Set or get the expected minimum string value.
- Property MinimumValue As String - Set or get the expected minimum value.
- Property NumTokens As Long - Number of string tokens in the tokenized response.
- Property ReferenceUnits As String - Set or get the comparison reference units.
- Property ReferenceValue As Double - Set or get the comparison reference value.
- Function SaveBlobCfg(hBlob As Long) As Long - Saves the configuration data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SaveBlobProp(hBlob As Long) As Long - Saves the property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created. Returns the hBlob if successful, 0 if failed.
- Function SendReceive() As Long - Send and/or receive communication data.
- Property StringMethod As STRING_TEST - Set or get the string test method.
- Property StringResponse As String - Get the string response from the UUT.
- Property StringTokens As String - Get the string tokens from the UUT.
- Property StringToSend As String - Set or get the string to send to the UUT.
- Property TestMethod As TEST_METHOD - Set or get the test method.
- Sub TRInit(hwndParent As Long, pdispHelperSvr As Object, pdispSPC As Object, bAlwaysFail As Long) - Initialization from the controlling test routine.
- Sub TRUnInit() - UnInitialization from the controlling test routine.
- Function UICfg() As Long - Display the configuration user interface. Returns TRUE if OK pressed.

- Function UIProp() As Long - Display the property user interface. Returns TRUE if OK pressed.
- Function UIVIP(pHWnd As Long) As Long - Display the basic VIP user interface.
- Function UIVIPDebug() As Long - Displays the debug VIP user interface.

5.28 Members of NHRINTERFACESLib.INHRWizardProgress

- NHR Wizard Progress Interface
- Sub ClientDescription(pStr As String) - Method ClientDescription
- Sub Decrement() - Method Decrement
- Sub EventDescription(pStr As String, EventIndex As Long) - Method EventDescription
- Function FeedBack(StepNumber As Long) As Long - Method FeedBack
- Sub Increment() - Method Increment
- Sub SetFeedBackObject(newObject As Object) - Method SetFeedBackObject
- Sub Show(hParent As Long) - Method Show
- Sub ShutDown() - Method ShutDown
- Sub StartPos(RightSide As Long, LeftSide As Long, Top As Long, Bottom As Long) - Method StartPos

5.29 Members of NHRINTERFACESLib.NHRBlob

- NHRBlob Class
- Function BlobLen(lHBlob As Long) As Long - Returns the length of a blob.
- Function BlobStruct(lHBlob As Long) As String - Returns a string describing the data structure for this blob.
- Function CreatorProgId(lHBlob As Long) As String - Returns a string of the ProgId of the creator of this blob.
- Sub DeleteBlob(lHBlob As Long) - Deletes a blob from the blob managers memory.
- Sub DeleteContents() - Deletes all blobs from the blob managers memory.
- Sub Dump(lHBlob As Long) - Dumps the blob at lHBlob. If lHBlob is 0, all blobs are dumped.
- Function FilenameStoredInBlob(lHBlob As Long) As String - Returns a string of the filename used to create this blob.
- Function HBlobFromIndex(lIndex As Long) As Long - Returns the hBlob for blob at specified index.
- Property Modified As Long - Modified flag to indicate data has been changed and not saved (read only).

- Property NumBlobs As Long - Number of blobs in the blob managers memory.
- Function OpenBlobFile(bstrPathName As String) As Long - Opens and loads blob file into the blob manager.
- Function RetrieveBlob(lHBlob As Long, pvntBlob) As Long - Retrieves a blob from the blob managers memory.
- Sub RetrieveFileFromBlob(lHBlob As Long, sFileToCreate As String) - Retrieves and creates a file from the blob managers memory.
- Function SaveBlobFile(bstrPathName As String) As Long - Member of NHRINTERFACESLib.NHRBlob - Saves the blobs currently in the blob manager to a file.
- Function Schema(lHBlob As Long) As Long - Member of NHRINTERFACESLib.NHRBlob - Returns the schema of a blob.
- Sub StoreBlob(plHBlob As Long, sCreatorProgId As String, sBlobStruct As String, lSchema As Long, pvntBlob, lLen As Long) - Member of NHRINTERFACESLib.NHRBlob - Stores the blob in the blob managers memory.
- Sub StoreFileAsBlob(plHBlob As Long, sFileToStore As String, lSchema As Long, sCreatorProgId As String) - Member of NHRINTERFACESLib.NHRBlob - Stores the file specified in the blob managers memory.

5.30 Members of NHRINTERFACESLib.NHRCanComm

- NHRCanComm Class
- Property BaudRate As Long - Set or get the baud rate.
- Property PortNumber As Long - Set or get the communication port number.

5.31 Members of NHRINTERFACESLib.NHRCfg

- NHRCfg Class
- Function AddChannel(bstrName As String) As Long - Add a channel to the configured list.
- Function AddDigital(bstrName As String, digType As DIGITAL_TYPE, lParentLogID As Long, lpunkParent As Unknown) As Long - Add a digital to the configured list.
- Function GetACPresenseSignalMasterDev(plLogID As Long) As Unknown - Get pointer to origin of AC Presense trigger.
- Function GetChannel(lIndex As Long) As String - Get channel name from configuration list by index number.
- Function GetChannelExternal(bstrName As String) As Long - TRUE if the channel is accessible to the user.
- Function GetChannelInUse(bstrName As String) As Long - TRUE if the channel has been used by a device.
- Function GetDCPresenseSignalMasterDev(plLogID As Long) As Unknown - Get pointer to origin of DC Presense trigger.

- Function GetDigital(lIndex As Long, pdigType As DIGITAL_TYPE) As String - Get digital name from configuration list by index number.
- Function GetDigitalInUse(bstrName As String) As Long - TRUE if the digital has been used by a device.
- Function GetDigitalParent(bstrName As String, plLogID As Long) As Unknown - Returns the parent of the digital. Note: perform an AddRef and Release if you hold onto this object.
- Function GetDigitalParentName(bstrName As String) As String - Returns the logical name of the parent of the digital.
- Function GetLogicalDev(pszLogDev As String, plLogID As Long) As Unknown - Get pointer to object and logical ID from logical name.
- Function GetMuxOwnership(plMuxOwnership As Long, [lWaitMS As Long = -1]) As Long - Get exclusive mux ownership rights.
- Function GetNumObjectsOfType(pszType As String) As Long - Get number of objects of specified type.
- Function GetObjectOfType(pszType As String, lIndex As Long, plLogID As Long) As Unknown - Get pointer to object of specified type.
- Function GetTypeOfObject(pszLogDev As String) As String - Get the type of the specified object.
- Sub Go() - Launches the visual configuration editor.
- Function Initialize(lpunkFaultMgr As Unknown) As Long - Performs vital component initialization. Must be called once after instantiation.
- Property Instance As Long - Number of active NHRCfg servers (read only).
- Property NumChannels As Long - Number of configured channels (read only).
- Property NumDigitals As Long - Number of configured digitals (read only).
- Property OffLine As Long - Get the offline state.
- Function ReleaseMuxOwnership(lMuxOwnership As Long) As Long - Release exclusive mux ownership rights.
- Sub RemoveChannel(bstrName As String) - Remove a channel from the configured list.
- Sub RemoveDigital(bstrName As String) - Remove a digital from the configured list.
- Sub ResetAll() - Reset all configured devices.
- Sub SetChannelExternal(bstrName As String, bExternal As Long) - TRUE if the channel is accessible to the user.
- Sub SetChannelInUse(bstrName As String, bInUse As Long) - TRUE if the channel has been used by a device.
- Sub SetDigitalInUse(bstrName As String, bInUse As Long) - TRUE if the digital has been used by a device.

- Sub SetLibBlobMgr(lpunkLibBlobMgr As Unknown) - Set the library blob manager property of all configured devices.
- Property SystemName As String - Get the system configuration name (read only).
- Function TestChannel(bstrName As String) As Long - Tests a channel name for uniqueness. Returns TRUE if the name is unique.
- Function TestDigital(bstrName As String) As Long - Tests a digital name for uniqueness. Returns TRUE if the name is unique.
- Function ValidateMuxOwnership(lMuxOwnership As Long) As Long - Validate exclusive mux ownership rights.

5.32 Members of NHRINTERFACESLib.NHRData

- NHRData Class
- Sub ActiveProgramDump([bPrinterFormat As Long = -1], psDump As String) - Creates a ascii dump of all properties for all objects in program. Set bPrinterFormat to TRUE for a printable dump; FALSE for a debug dump.
- Property ActiveProgramEnableMultiFixture As Long - Non 0 if using an A/B fixture.
- Property ActiveProgramFixtureSwitchDelay As Double - Delay after switching ActiveProgramFixtureSwitchDout.
- Property ActiveProgramIndex As Long - Active program index (-1 if none).
- Property ActiveProgramName As String - Active program name (blank if none).
- Property ActiveProgramPreferences As Long - Handle to the preferences blob.
- Property ActiveProgramTestInfoCfg As Long - Handle to the cfg blob.
- Property ActiveProgramTestInfoProgId As String - Active program test info object progid (blank if none).
- Property BlobMgr As Object - Pointer to the blob manager.
- Function CreateProgram(lProgramIndex As Long, bRelated As Long, sProgramName As String, sProgramVersion As String, sComments As String) As Long - Create a new program. If lProgramIndex < 0 create a new empty program. If lProgramIndex >= 0 then create a new program based on that index. bRelated is TRUE to make new related to lProgramIndex.
- Sub DestroyProgram(lProgramIndex As Long) - Destroys a program.
- Function GetActiveProgramFixtureBusyDout(lFixtureIndex As Long) As String - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Function GetActiveProgramFixtureFailDout(lFixtureIndex As Long) As String - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Function GetActiveProgramFixtureGoDin(lFixtureIndex As Long) As String - Logical name of the I/O for the fixture specified by lFixtureIndex.

- Function GetActiveProgramFixturePassDout(lFixtureIndex As Long) As String - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Function GetActiveProgramFixtureReadyDout(lFixtureIndex As Long) As String - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Function GetActiveProgramFixtureSwitchDout(lFixtureIndex As Long, lOutputIndex As Long) As String - Logical name of the Dout at lOutputIndex (0 or 1) used to make lFixtureIndex (0 or 1) active.
- Sub GetProgramInfo(lProgramIndex As Long, psProgramName As String, psProgramVersion As String, psCreatedBy As String, pdateCreated As Date, psCreatedVersion As String, psModifiedBy As String, pdateModified As Date, psModifiedVersion As String, psComments As String) - Get the program info.
- Sub GetProgramName(lProgramIndex As Long, psProgramName As String) - Get the program name.
- Property IsModified As Long - TRUE if any program in the library has been modified. Set to FALSE by Load and SaveLibrary.
- Property LibraryPath As String - The full path to the currently loaded library (blank if none).
- Function LoadLibrary(sFilename As String) As Long - Load the program library from disk.
- Sub Logon(sName As String, sVersion As String, punkCfgMgr As Unknown, pdispFaultMgr As Object) - Logon to the data manager. Nothing can be done without logging on.
- Sub LogonReadOnly(punkCfgMgr As Unknown, pdispFaultMgr As Object) - Logon for read-only to the data manager. Nothing can be done without logging on.
- Sub NewLibrary() - Creates a new (empty) program library.
- Property NumPrograms As Long - Number of programs in the library (read only).
- Function SaveLibrary(sFilename As String) As Long - Saves the program library to disk.
- Sub SetActiveProgramFixtureBusyDout(lFixtureIndex As Long, sFixtureBusyDout As String) - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Sub SetActiveProgramFixtureFailDout(lFixtureIndex As Long, sFixtureFailDout As String) - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Sub SetActiveProgramFixtureGoDin(lFixtureIndex As Long, sFixtureGoDin As String) - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Sub SetActiveProgramFixturePassDout(lFixtureIndex As Long, sFixturePassDout As String) - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Sub SetActiveProgramFixtureReadyDout(lFixtureIndex As Long, sFixtureReadyDout As String) - Logical name of the I/O for the fixture specified by lFixtureIndex.
- Sub SetActiveProgramFixtureSwitchDout(lFixtureIndex As Long, lOutputIndex As Long, sFixtureSwitchDout As String) - Logical name of the Dout at lOutputIndex (0 or 1) used to make lFixtureIndex (0 or 1) active.

- Sub SetProgramInfo(lProgramIndex As Long, sProgramName As String, sProgramVersion As String, sComments As String) - Set the program info.
- Sub SetProgramName(lProgramIndex As Long, sProgramName As String) - Set the program info.

5.33 Members of NHRINTERFACESLib.NHRDatalog

- NHRDatalog Class
- Function Label(lIndent As Long, sLabel As String, bShowAsFault As Long, lParentGroup As Long, lSortOrder As Long) As Long - Log a Label.
- Sub LabelImm(lIndent As Long, sLabel As String, bShowAsFault As Long) - Method LabelImm.
- Function StrResult(lIndent As Long, sLabel As String, sdMinimum As String, sActual As String, sMaximum As String, sResult As String, lResultType As Long, lParentGroup As Long, lSortOrder As Long) As Long - Log a Result with text data.
- Sub StrResultImm(lIndent As Long, sLabel As String, sdMinimum As String, sActual As String, sMaximum As String, sResult As String, lResultType As Long) - Method StrResultImm.
- Function ValResult(lIndent As Long, sLabel As String, dMinimum As Double, dActual As Double, dMaximum As Double, sUnits As String, sResult As String, lResultType As Long, lParentGroup As Long, lSortOrder As Long) As Long - Log a Result with numeric data.
- Sub ValResultImm(lIndent As Long, sLabel As String, dMinimum As Double, dActual As Double, dMaximum As Double, sUnits As String, sResult As String, lResultType As Long) - Method ValResultImm.

5.34 Members of NHRINTERFACESLib.NHRDigitalContainer

- NHRDigitalContainer Class
- Function GetBit(bstrBit As String) As DIGITAL_STATE - Get the state of the digital bit.
- Property NumBits As Long - Get the number digital bits in the container (read only).
- Function SetBit(bstrBit As String, lState As DIGITAL_STATE, [bWaitForComplete As Long]) As Long - Set the state of the digital bit.

5.35 Members of NHRINTERFACESLib.NHRFault

- NHRFault Class
- Sub AddContext(sAdditionalInfo As String) - Adds a context string to the error information already stored.
- Function CheckFault() As Long - Returns TRUE if a fault has been stored.
- Sub ClearFault() - Clears the fault. Used after the fault has been resolved.

- Sub DisplayFault() - Displays the stored fault information in a message box, and clears the stored fault.
- Function GetFault(plSecondaryFault As Long, pbstrFaultDescription As String, pbstrAdditionalInfo As String, pdDate As Date) As Long - Returns the fault descriptions, time, and fault code of the stored fault. Does not clear the stored fault.
- Sub Initialize() - Initializes NHRFault's external programs.
- Function SetFault(lFault As Long, [lSecondaryFault As Long]) As Long - Evaluates lFault and, if necessary, stores it (or adds it as additional context if a fault pre-exists). Returns TRUE if a fault exists in this thread.

5.36 Members of NHRINTERFACESLib.NHRHelper

- NHRHelper Class
- Property CfgSvr As Unknown - Set to the configuration server object instance used for config data (write only).
- Property DataMgr As Unknown - Set to the data manager object instance (write only).
- Function EvaluateStrResult(lStringTestType As STRING_TEST, bCaseSensitive As Long, sMinimum As String, psString As String, sMaximum As String, psResult As String, psTestDescription As String, [bAlwaysFail As Long], [pIDispatchSPC As Object]) As RESULT_TYPE - Compare the string against a minimum and maximum and return the result as a RESULT_TYPE.
- Function EvaluateValResult(dLowerExtreme As Double, dMinimum As Double, pdValue As Double, dMaximum As Double, dUpperExtreme As Double, psResult As String, [bAlwaysFail As Long], [pIDispatchSPC As Object]) As RESULT_TYPE - Compare the value against a minimum and maximum and return the result as a RESULT_TYPE. The "Extreme" values are used for offline data distribution.
- Function GetRefCount(punk As Unknown) As Long - Get any objects current reference count
- Sub GetSysControl(scStatus As SC_STATUS) - Check the sys control buttons and multi-fixture inputs.
- Sub InitSysControl() - Prepare to use the sys control lights and multi-fixture signals.
- Property IsInputOn As Long - Input state is ON if TRUE (read only).
- Function SetAnalogOff() As Long - Turn off every analog output.
- Function SetAnalogVector(lAnalogVectorId As Long) As Long - Set each analog output according to the supplied vector ID.
- Function SetDigitalOff() As Long - Turn off every digital output.
- Function SetDigitalVector(lDigitalVectorId As Long) As Long - Set each digital output according to the supplied vector ID.

- Function SetInputVector(lMode As INPUT_VECTOR_MODE, lInputVectorId As Long, [bReversed As Long]) As Long - Set each input according to the supplied vector ID. Set bReversed to TRUE to set in reverse order.
- Function SetOutputVector(lOutputVectorId As Long) As Long - Set each output according to the supplied vector ID.
- Function SetOutputVectorForceCR(lOutputVectorId As Long) As Long - Set each output according to the supplied vector ID except force to CR Mode.
- Sub SetSysControl(lFixtureIndex As Long, scState As SC_STATE) - Set the sys control lights and multi-fixture signals.
- Sub SynchronizeCommands() - Wait for all communication to all devices to be completed before continuing.
- Property TriggerOnNext As Long - TRUE to generate a trigger when the next vector is set.

5.37 Members of NHRINTERFACESLib.NHRInputContainer

- NHRInputContainer Class
- Property CurrentLimitMax As Double - Maximum programmable current limit, in Amps (read only).
- Property CurrentLimitMin As Double - Minimum programmable current limit, in Amps (read only).
- Property CurrentRange As Double - Set or get the operating current range of the input, in Amps.
- Property CurrentRangeStr As String - Set or get the operating current range of the input in string form, as a value or percentage of reference current.
- Property FrequencyMax As Double - Maximum programmable frequency, in Hertz (read only).
- Property FrequencyMin As Double - Minimum programmable frequency, in Hertz (read only).
- Function GetCurrentLimit([lPhase As Long]) As Double - Get the programmed current limit of the input, in amps.
- Function GetCurrentLimitStr([lPhase As Long]) As String - Get the programmed current limit of the input in string form, as a value or percentage of reference current.
- Function GetFrequency() As Double - Get the programmed frequency of the input, in Hertz.
- Function GetFrequencyStr() As String - Get the programmed frequency of the input in string form, as a value or percentage of reference frequency.
- Function GetPhaseControl() As PHASE_CONTROL - Get the active phase mode and control of the input.
- Function GetVoltage([lPhase As Long]) As Double - Get the programmed voltage of the input, in volts.

- Function GetVoltageStr([lPhase As Long]) As String - Get the programmed voltage of the input in string form, as a value or percentage of reference voltage.
- Property IsFreqProgrammable As Long - TRUE if frequency is programmable (read only).
- Property IsPhaseControllable As Long - TRUE if phases can be individually controlled (read only).
- Property NumPhases As Long - How many phases this input supports (read only).
- Property RefCurrent As Double - Set or get the reference current, in Amps.
- Property RefCurrentStr As String - Set or get the reference current in string form.
- Property RefFrequency As Double - Set or get the reference frequency, in Hertz.
- Property RefFrequencyStr As String - Set or get the reference frequency in string form.
- Property RefTime As Double - Set or get the reference time, in Seconds.
- Property RefTimeStr As String - Set or get the reference time in string form.
- Property RefVoltage As Double - Set or get the reference voltage, in Volts.
- Property RefVoltageStr As String - Set or get the reference voltage in string form.
- Property RefWatts As Double - Set or get the reference wattage, in Watts.
- Property RefWattsStr As String - Set or get the reference wattage in string form.
- Property SequenceDelay As Double - Set or get the sequence delay of the input, in Seconds.
- Property SequenceDelayStr As String - Set or get the sequence delay of the input in string form, as a value or percentage of reference time.
- Function SetCurrentLimit(dAmps As Double, [lPhase As Long], [bWaitForComplete As Long]) As Long - Set the programmed current limit of the input, in amps.
- Function SetCurrentLimitStr(bstrAmps As String, [lPhase As Long], [bWaitForComplete As Long]) As Long - Set the programmed current limit of the input in string form, as a value or percentage of reference current.
- Function SetFrequency(dFrequency As Double, [bWaitForComplete As Long]) As Long - Set the programmed frequency of the input, in Hertz.
- Function SetFrequencyStr(bstrFreq As String, [bWaitForComplete As Long]) As Long - Set the programmed frequency of the input in string form, as a value or percentage of reference frequency.
- Function SetPhaseControl(lControl As PHASE_CONTROL, [bWaitForComplete As Long]) As Long - Set the active phase mode and control of the input.
- Function SetVoltage(dVolts As Double, [lPhase As Long], [bWaitForComplete As Long]) As Long - Set the programmed voltage of the input, in volts.

- Function SetVoltageStr(bstrVolts As String, [lPhase As Long], [bWaitForComplete As Long]) As Long - Set the programmed voltage of the input in string form, as a value or percentage of reference voltage.
- Property VoltageMax As Double - Maximum programmable voltage, in Volts (read only).
- Property VoltageMin As Double - Minimum programmable voltage, in Volts (read only).
- Property VoltageRange As Double - Set or get the operating voltage range of the input, in Volts.
- Property VoltageRangeStr As String - Set or get the operating voltage range of the input in string form, as a value or percentage of reference voltage.
- Function WaitSequenceDelay() As Long - Wait the amount of time specified by the SequenceDelay property.
- Property WattsMax As Double - Maximum programmable wattage, in Watts (read only).
- Property WattsMin As Double - Minimum programmable wattage, in Watts (read only).

5.38 Members of NHRINTERFACESLib.NHRLogOn

- NHR LogOn Interface
- Property AccessValue As Long - Property AccessValue
- Function Admin() As Long - Method Admin
- Property IDKey As Long - Property IDKey
- Function Logon() As Long - Method LogOn
- Property UserLevel As String - Property UserLevel
- Property UserName As String - Property UserName
- Members of NHRINTERFACESLib.NHROutputContainer
- NHROutputContainer Class
- Property Channel As String - Set or get the measurement logical channel name.
- Property CurrentMax As Double - Maximum programmable current, in Amps (read only).
- Property CurrentMin As Double - Minimum programmable current, in Amps (read only).
- Property CurrentRange As Double - Set or get the operating current range of the input, in Amps.
- Property CurrentRangeStr As String - Set or get the operating current range of the input in string form, as a value or percentage of reference current.
- Function FindTransientDev(dTransient As Double, plLogID As Long) As Unknown - Returns a pointer to IUnknown of a device capable of performing the transient.
- Function GetMode() As LOAD_MODES - Get the programmed mode of the output, in numeric form.

- Function GetModeStr([bVerbose As Long]) As String - Get the programmed mode of the output in string form.
- Function GetValue() As Double - Get the programmed value of the output. Units are dependent on the output mode.
- Function GetValueStr() As String - Get the programmed value of the output in string form. Units are dependent on the output mode.
- Property NomVoltage As Double - Set or get the nominal voltage, in Volts.
- Property NomVoltageStr As String - Set or get the nominal voltage in string form.
- Property RefCurrent As Double - Set or get the reference current, in Amps.
- Property RefCurrentStr As String - Set or get the reference current in string form.
- Property RefFrequency As Double - Set or get the reference frequency, in Hertz.
- Property RefFrequencyStr As String - Set or get the reference frequency in string form.
- Property RefResistance As Double - Set or get the reference resistance, in Ohms.
- Property RefResistanceStr As String - Set or get the reference resistance in string form.
- Property RefTime As Double - Set or get the reference time, in Seconds.
- Property RefTimeStr As String - Set or get the reference time in string form.
- Property RefVoltage As Double - Set or get the reference voltage, in Volts.
- Property RefVoltageStr As String - Set or get the reference voltage in string form.
- Property RefWatts As Double - Set or get the reference wattage, in Watts.
- Property RefWattsStr As String - Set or get the reference wattage in string form.
- Property ResistanceMax As Double - Maximum programmable resistance, in Ohms (read only).
- Property ResistanceMin As Double - Minimum programmable resistance, in Ohms (read only).
- Function SetMode(lMode As LOAD_MODES, [bWaitForComplete As Long]) As Long - Set the programmed mode of the output, in numeric form.
- Function SetModeStr(bstrMode As String, [bWaitForComplete As Long]) As Long - Set the programmed mode of the output in string form.
- Function SetValue(dValue As Double, [bWaitForComplete As Long]) As Long - Set the programmed value of the output. Units are dependent on the output mode.
- Function SetValueStr(bstrValue As String, [bWaitForComplete As Long]) As Long - Set the programmed value of the output in string form. Units are dependent on the output mode.
- Property VoltageMax As Double - Maximum programmable voltage, in Volts (read only).
- Property VoltageMin As Double - Minimum programmable voltage, in Volts (read only).

- Property VoltageRange As Double - Set or get the operating voltage range of the input, in Volts.
- Property VoltageRangeStr As String - Set or get the operating voltage range of the input in string form, as a value or percentage of reference voltage.
- Property WattsMax As Double - Maximum programmable wattage, in Watts (read only).
- Property WattsMin As Double - Minimum programmable wattage, in Watts (read only).

5.39 Members of NHRINTERFACESLib.NHRSerial

- NHRSerial Class
- Property Address As Byte - Device address. (Read-Write)
- Property BusyBitMask As Byte - Bits of the status register indicating busy when set (write only).
- Property DelayBeforeFirstStatus As Double - Normal delay before status is check. (Read-Write)
- Property DelayBetweenStatus As Double - Delay while status is busy. (Read-Write)
- Function GetResp(lClaimCheck As Long, psaCmd) As Long - Retrieves a response from the port (read only).
- Function IsProcessed(lClaimCheck As Long) As Long - TRUE if a command has been completed (read only).
- Sub OpenPort(pbSuccess As Long) - Opens the port for serial communication.
- Sub OpenPortForTest(lDevID As Long, pbSuccess As Long) - Opens the port for testing serial communication.
- Property ReadyBitMask As Byte - Bits of the status register indicating ready when set (write only).
- Function RemoveStatusFromHistory(pdwStatus As <Unsupported variant type>) As Long - Removes the oldest status from the list. Returns False if the list if empty, else True with a valid status.
- Sub RemoveStatusFromHistoryC(pdwStatus As <Unsupported variant type>) - Removes the oldest status from the list. Returns False if the list if empty, else True with a valid status.
- Function SendCmd(lCmdType As Long, psaCmd, lCmdLength As Long, bWait As Long) As Long - Sends a command to the port (write only).
- Property TimeOut As Double - Wait time for a type of command to complete. (Read-Write)

5.40 Members of NHRINTERFACESLib.NHRSerialComm

- NHRSerialComm Class
- Property BaudRate As Long - Set or get the baud rate.
- Property ByteSize As Long - Set or get the number of bits per byte.
- Function CommExtendedFunction(exFunction As SERIAL_EXTENDED_FUNCTION) As Long - Perform an extended communication function.
- Property DeviceControl As Long - Set or get the device control word.
- Property EOFChar As String - Set or get the end of file character.
- Property ErrorChar As String - Set or get the error replacement character.
- Property EventChar As String - Set or get the received event character.
- Property InBufferSize As Long - Set or get the input buffer size.
- Property OutBufferSize As Long - Set or get the output buffer size.
- Property Parity As SERIAL_PARITY - Set or get the parity of the communication port.
- Property PnPDeviceID As String - Get the plug-n-play External COM Device ID string.
- Property PortNumber As Long - Set or get the communication port number.
- Property ReadIntervalTimeout As Double - Set or get the maximum time allowed between the arrival of two characters.
- Property ReadTOConstant As Double - Set or get the read timeout constant.
- Property ReadTOMultiplier As Double - Set or get the read timeout multiplier.
- Property StopBits As SERIAL_STOPBITS - Set or get the number of stop bits.
- Property WriteTOConstant As Double - Set or get the write timeout constant.
- Property WriteTOMultiplier As Double - Set or get the write timeout multiplier.
- Property XOFFChar As String - Set or get the transmit and receive XOFF character.
- Property XOFFLimit As Long - Set or get the transmit XOFF threshold.
- Property XONChar As String - Set or get the transmit and receive XON character.
- Property XONLimit As Long - Set or get the transmit XON threshold.

6. CUSTOM ROUTINE TEMPLATE

Use this procedure to create a new *em**Power*** ® *Test Routine* based upon the provided template files. In the steps that follow, substitute your test routine name for every occurrence of `MyTestRoutine`.

⇨ Copy the template files to a new directory

⇨ Rename the files as follows:

`NHRTRTemplate.bas` → `MyTestRoutine.bas`

`NHRTRTemplate.cls` → `MyTestRoutine.cls`

`NHRTRTemplateSvr.vbp` → `MyTestRoutineSvr.vbp`

Using a document editor such as *Notepad* or *Wordpad*:

⇨ Edit the file `MyTestRoutineSvr.vbp`, replacing every occurrence of `NHRTRTemplate` with `MyTestRoutine`

These will not be whole word replacements – `NHRTRTemplateMain` becomes `MyTestRoutineMain`. Be sure to save this as a text document if the editor asks.

Using a document editor:

⇨ Edit `MyTestRoutine.bas`, replacing every occurrence of `NHRTRTemplate` with `MyTestRoutine`

Again, these will not be whole word replacements.

Using a document editor:

⇨ Edit `MyTestRoutine.cls`, replacing every occurrence of `NHRTRTemplate` with `MyTestRoutine`

⇨ Load the test routine project into Visual Basic by either double-clicking on `MyTestRoutineSvr.vbp`, or launching VB and browsing for the project file.

In the "Project Explorer" window (Ctrl+R):

⇨ Expand the "Module" entry

⇨ Double-click the module named `MyTestRoutineMain` to load it into the VB editor.

⇨ Modify the part of the `Public Const sNHRTRName` line which reads "`Name This Test`" to read "`My Test Routine`"

This entry is displayed in the PowerEdit "Test Routine" column tree view. You may optionally modify the category descriptors to control the placement in the tree control.

- ⇨ Modify the value of the `Public Const sNHRTRBlobStruct` line to read "`MyTestRoutineBlob`"
- ⇨ Replace (Ctrl+H) every occurrence of `TemplateProp` with `MyTestRoutineProp` in the entire module

To modify the version information:

- ⇨ Select "Project" from the main menu, then
- ⇨ "MyTestRoutine properties…"

At the "General" tab:

- ⇨ Modify the "Project Description" field as desired.

At the "Make" tab:

- ⇨ Modify the "Version Number" and "Version Information" fields as desired.

You are now ready to modify the test program as necessary to perform your specific task.

Places to change in the Template:

- ⇨ Modify the form (`frmUIProp.frm`) as required

In the *Modules* section:

- ⇨ Open the code window of: `MyTestRoutine.bas`
- ⇨ Find the *comment*: `'Template test properties`
- ⇨ Add your *Properties* to the type declaration
- ⇨ Delete: `bDoTest As Long` if you don't need it
- ⇨ Find the *sub-procedure*: `InitializePropBlob()`
- ⇨ Assign default values for your *Blob Properties* – you may choose to use a `With` block to do this

```
With g_blobProp
    .lSource = 0
    .dFinal = 0
    .lDelay = 0
    .dSize = 0
End With
```

- ⇨ Open the form (`frmUIProp.frm`)
- ⇨ Double-click the form – the code window opens inside the *sub-procedure* `Form_Load()`
- ⇨ Assign values from the *Blob* into your form properties – you may choose to use a `With` block to do this

- ⇨ Find the *sub-procedure*: `SaveProperties()`
- ⇨ Add code to copy the values of the form's properties into the *Blob*
- ⇨ Open the class module
- ⇨ Find the *Private Function*: `INHRTR_GO()`
- ⇨ Add your code
- ⇨ Find the *Function*: `INHRTR_DumpProp()`
- ⇨ Find the *comment*: `'Dump your test properties here`
- ⇨ Add code to make the string using text to describe the variable and values from the *Blob*

GLOSSARY

COM - *C*omponent *O*bject *M*odel. An architecture for defining interfaces and interaction among objects implemented by widely varying software applications. Do not confuse this with COMM, the typical abbreviation for a Serial port.

GUID - *G*lobally *U*nique *Id*entifier. A 16-byte (128-bit) value that uniquely identifies some entity, such as a COM class, or COM interface. A GUID is also referred to as a Universally Unique Identifier (UUID).

BLOB - *B*inary *L*arge *Ob*ject. The name of the data structures used to store test routine, device driver and other configuration information in *em**Power***®.