*em**Power*** ®

Application Programming Interface

Reference Manual

Part No. 09-0276

REVISION B

April 10, 2002

Document History

Rev A	**ECO 15390**	**9/05/01**	**B. H.**
Rev B	**ECO 15505**	**4/10/02**	**B. L.**

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.

TABLE OF CONTENTS

1. MEMBER OF NHRINTERFACESLIB.INHRCONTAINER

1.1 Describe (Container)

Sets or gets a user defined description of the container.
Member of NHRINTERFACESLib.INHRContainer

Property: Describe As String

Parameters:

Sets the user defined description of the Container

Return Values:

Returns the user defined description of the Container

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Containers you must obtain a reference to either the **INHRDataOutput**, **INHRDataInput**, or **INHRDataDigital** interface of the Data Manager. Use the **NumOutContainers**, **NumInContainers**, or **NumDigitalContainers** properties, respectively, to determine the number of Containers available.

Example:

Obtain a reference to the first Input Container in the program, get the description and then properly release the object:

```
Dim lIDContainer As Long
Dim obContainer As INHRContainer
Dim sName As String

lIDContainer = 0
Set obContainer = obDataMgrIn.GetInContainerObj(lIdContainer)
sName = obContainer.Describe
Set obContainer = Nothing
```

See Also:

NumOutContainers, NumInContainers, NumDigitalContainers, GetInContainerObj

2. MEMBER OF NHRINTERFACESLIB.INHRDATAANALOG

2.1 GetAnalogVectorLabel

Gets the label for the vector id.
Member of NHRINTERFACESLib.INHRDataAnalog

Function: **GetAnalogVectorLabel (**
lIDVector As Long
) As String

Parameters:

lIDVector:

The record ID of the desired digital vector.

Return Values:

Returns the contents of the Vector Name field describing the vector.

Remarks:

The Data Manager maintains the list of all analog vectors defined in the program. To gain access to analog vectors you must obtain a reference to the **INHRDataAnalog** interface of the Data Manager. Use the **GetNumAnalogVectors** function to determine the number of analog vectors available, and **GetAnaloglVectorId** to obtain the record ID of each vector.

The Vector Name column is located on the PowerEdit Analog pane.

Example:

Iterate through each analog vector in the program and obtain the vector name of each analog vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrAnalog.GetNumAnalogVectors - 1
    lVectorID = obDataMgrAnalog.GetAnalogVectorId(lCount)
    sVectorName = obDataMgrAnalog.GetAnalogVectorLabel(lVectorID)
Next
```

See Also:

GetNumAnalogVectors, GetAnalogVectorId

2.2 GetNumAnalogVectors

The number of analog control vectors.
Member of NHRINTERFACESLib.INHRDataAnalog

Function: **GetNumAnalogVectors () As Long**

Parameters:

None

Return Values:

Returns the number of analog vectors configured in the test program.

Remarks:

The Data Manager maintains the list of all analog vectors defined in the program. To gain access to analog vectors you must obtain a reference to the INHRDataAnalog interface of the Data Manager.

Example:

Iterate through each analog vector in the program and obtain the vector name of each analog vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrAnalog.GetNumAnalogVectors - 1
    lVectorID = obDataMgrAnalog.GetAnalogVectorId(lCount)
    sVectorName = obDataMgrAnalog.GetAnalogVectorLabel(lVectorID)
Next
```

See Also:

GetAnalogVectorId, GetAnalogVectorLabel

3. MEMBER OF NHRINTERFACESLIB.NHRDATALOG

3.1 Label

Log a label.
Member of NHRINTERFACESLib.NHRDatalog

Function: **Label (**
lIndent As Long,
sLabel As String,
bShowAsFault as Long,
lParentGroup As Long,
lSortOrder As Long
) As Long

Parameters:

lIndent:

The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

slabel:

A string defining the value being logged. This value is normally placed in the "Label" column of the datalog device.

bShowAsFault:

True to cause the dataloggers to format the **sLabel** string as appropriate to a fault message.

lParentGroup:

Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

Return Values:

Returns a new Parent Group index that can be used as the lParentGroup parameter in subsequent datalog calls.

Remarks:

This command logs the supplied label strings to all the configured dataloggers. This command is used to log informational text to the dataloggers. This informational text may be interpreted as a fault message if the **bShowAsFault** parameter is True. Note: logging a fault does not cause the routine or the UUT to fail; it simply informs the dataloggers to format the string as appropriate to a fault message.

Within the same test routine, multiple values logged with the same **lParentGroup** number are grouped together and sub-sorted according to **lSortOrder**. In the case of duplicate **lSortOrder** values, the values are logged in the order they were received. You may pass zero values for both sorting parameters to log all measurement in the order they were received. **lParentGroup** and **lSortOrder** parameters are not maintained across separate test routine executions. Refer to the example under "ValResult".

Example:

Log a Fault to the dataloggers:

```
Dim lFault As Long
Dim lSecondary As Long
Dim sDescription As String
Dim sMoreInfo As String
```

```
Dim dateCode As Date

If g_obFaultMgr.CheckFault Then
    lFault = g_obFaultMgr.GetFault(lSecondary, sDescription, sMoreInfo, dateCode)
    g_obFaultMgr.ClearFault
    g_obDatalog.Label 0, "Error: " & sDescription & "; " & sMoreInfo, True, 0, 0
    m_lFailCount = m_lFailCount + 1
End If
```

Sample Screen driver output:

Label	Minimum	Actual	Maximum	Result

Error: NHR Invalid Parameter.; AC Source 1: SetVoltage: Illegal source voltage: 500

See Also:

CheckFault, GetFault, ClearFault, StrResult, ValResult.

3.2 StrResult

Log a Result with numeric data.
Member of NHRINTERFACESLib.NHRDatalog

Function: **StrResult (**
lIndent As Long,
sLabel As String,
sMinimum As String,
sActual As String,
sMaximum As String,
sResult As String,
lResultType As Long,
lParentGroup As Long,
lSortOrder As Long
) As Long

Parameters:

lIndent:

The indent level for the provided label. An indent level is equivalent to one tab stop of the device.

sLabel:

A string defining the value being logged. This value is normally placed in the "Label" column of the datalog device. May be an empty string if no label is desired.

sMinimum:

The minimum acceptable string for the measurement being logged. This string is normally placed in the "Minimum" column of the datalog device. May be an empty string if no minimum string is desired

sActual:

The string value being logged. This string is normally placed in the "Actual" column of the datalog device.

sMaximum:

The maximum acceptable string for the measurement being logged. This string is normally placed in the "Maximum" column of the datalog device. May be an empty string if no maximum string is desired.

sResult:

A string representation of the results of the measurement evaluation, usually either "Pass", "Fail" or "Untested". This string is normally placed in the "Result" column.

lResultType:

The results of the measurement evaluation. This must be one of the RESULT_TYPE enumeration values. Datalog drivers use this value when determining the pass or fail state of this measurement.

lParentGroup:

Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

Return Values:

Returns a new Parent Group index that can be used as the **lParentGroup** parameter in subsequent datalog calls.

Remarks:

This command logs the supplied data strings to all the configured dataloggers. The drivers use the **lResultType** parameter to determine the pass/fail state of the measurement, and not the **sResult** parameter. The result may or may not be written to the datalog device depending on the configuration of the datalog driver.

Within the same test routine, multiple values logged with the same **lParentGroup** number are grouped together and sub-sorted according to **lSortOrder**. In the case of duplicate **lSortOrder** values, the values are logged in the order they were received. You may pass zero values for both sorting parameters to log all measurement in the order they were received. **lParentGroup** and **lSortOrder** parameters are not maintained across separate test routine executions. Refer to the example under "ValResult".

Example:

Send a string to the dataloggers:

```
Dim sMeasurement As String

SMeasurement = "123ABC"
g_obDatalog.StrResult 1, "UUT Communications", "ABC123", sMeasurement, "", "Fail",
FAIL_RESULT, 0, 0
```

Sample Screen driver output:

Label	Minimum	Actual	Maximum	Result
UUT Communications	ABC123	123ABC		Fail

See Also:

EvaluateStrResult, ValResult

3.3 ValResult

Log a Result with numeric data.
Member of NHRINTERFACESLib.NHRDatalog

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

Parameters

lIndent:

The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

sLabel:

A string defining the value being logged. This value is normally placed in the “Label” column of the datalog device. May be an empty string if no label is desired.

dMinimum:

The minimum acceptable value for the measurement being logged. This value is normally placed in the “Minimum” column of the datalog device.

dActual:

The numeric value being logged. This value is normally placed in the “Actual” column of the datalog device.

dMaximum:

The maximum acceptable value for the measurement being logged. This value is normally placed in the “Maximum” column of the datalog device.

sUnits:

The units of the value being logged (e.g., “V”, “Hz”, etc.) May be an empty string if no units are desired.

sResult:

A string representation of the results of the measurement evaluation, usually either “Pass”, “Fail” or “Untested”. This string is normally placed in the “Result” column.

lResultType:

The results of the measurement evaluation. This must be one of the RESULT_TYPE enumeration values. Datalog drivers use this value when determining the pass or fail state of this measurement.

lParentGroup:

Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

Return Values:

Returns a new Parent Group index that can be used as the **lParentGroup** parameter in subsequent datalog calls.

Remarks:

This command logs the supplied data values to all the configured dataloggers. The drivers use the **lResultType** parameter to determine the pass/fail state of the measurement, and not the **sResult** parameter. The result may or may not be written to the datalog device depending on the configuration of the datalog driver.

Within the same test routine, multiple values logged with the same **lParentGroup** number are grouped together and sub-sorted according to **lSortOrder**. In the case of duplicate **lSortOrder** values, the values are logged in the order they were received. You may pass zero values for both sorting parameters to log all measurement in the order they were received. **lParentGroup** and **lSortOrder** parameters are not maintained across separate test routine executions.

Example:

The following example illustrates the effect of **lParentGroup** and **lSortOrder**. Assume that the first measurement cycle gathers voltage measurements and the second gathers current measurements but you desire the voltage and current measurements for each channel to be logged next to each other.

Since the **lParentGroup** and **lSortOrder** values of the three grouping labels are zero, they become root entries, sorted in the order they are datalogged (see Sample Screen driver output). The "Volts" and "Amps" values are logged with the return value of these grouping labels as the **lParentGroup** values, making them children of the corresponding label. Since the "Volts" **lSortOrder** value is less than the "Amps" sort order, "Volts" appears first in the datalog:

```
Dim dMeasurement(0 To 2) As Double
Dim i As Integer
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lGroup(0 To 2) As Long

lGroup(0) = g_obDatalog.Label(0, "Channel 1", False, 0, 0)
lGroup(1) = g_obDatalog.Label(0, "Channel 2", False, 0, 0)
lGroup(2) = g_obDatalog.Label(0, "Channel 3", False, 0, 0)

obDMM.SetupBlobs lLogID, sMuxChannel, hBlob, dRange, 3
obDMM.Meas lLogID
obDMM.Fetch lLogID, dMeasurement(0), 3

For i = 0 To 2
    lResult = g_obHelper.EvaluateValResult(DBL_MIN, 4.95, dMeasurement, 5.05,
DBL_MAX, sResult)
    g_obDatalog.ValResult 1, "Volts", 4.95, dMeasurement(i), 5.05, "V", sResult,
lResult, lGroup(i), 0
Next

obDMM.SetupBlobs lLogID, sMuxChannel, hBlob2, dRange, 3
obDMM.Meas lLogID
obDMM.Fetch lLogID, dMeasurement(0), 3

For i = 0 To 2
    lResult = g_obHelper.EvaluateValResult(DBL_MIN, 1.1, dMeasurement, 2.5,
DBL_MAX, sResult)
    g_obDatalog.ValResult 1, "Amps", 1.1, dMeasurement(i), 2.5, "A", sResult,
lResult, lGroup(i), 1
Next
```

Sample Screen driver output:

Label	Minimum	Actual	Maximum	Result
Channel 1				
Volts	4.950 V	4.997 V	5.050 V	Pass
Amps	1.100 A	1.736 A	2.500 A	Pass
Channel 2				

```
        Volts          4.950 V      5.025 V      5.050 V      Pass
        Amps           1.100 A      1.583 A      2.500 A      Pass
    Channel 3
        Volts          4.950 V      4.967 V      5.050 V      Pass
        Amps           1.100 A      2.024 A      2.500 A      Pass
```

See Also:

SetupBlobs, Meas, Fetch, EvaluateValResult

4. MEMBER OF NHRINTERFACESLIB.INHRDATADIGITAL

4.1 GetDigitalVectorId

Returns the id for the digital vector at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataDigital

Function: **GetDigitalVectorId (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired vector.

Return Values:

Returns the record ID of the digital vector indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager maintains the list of all digital vectors defined in the program. To gain access to digital vectors you must obtain a reference to the INHRDataDigital interface of the Data Manager. Use the GetNumDigitalVectors function to determine the number of digital vectors available, and GetDigitalVectorId to obtain the record ID of each vector.

Example:

Iterate through each digital vector in the program and obtain the vector name of each digital vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrDigital.GetNumDigitalVectors - 1
    lVectorID = obDataMgrDigital.GetDigitalVectorId(lCount)
    sVectorName = obDataMgrDigital.GetDigitalVectorLabel(lVectorID)
Next
```

See Also:

GetNumDigitalVectors, GetDigitalVectorLabel

4.2 GetDigitalContainerId

Returns the id for the container at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataDigital

Function: **GetDigitalContainerId (**
lIndex As Long
) As Long

Parameters:

lIndex:

The zero-based index of the desired Digital Container.

Return Values:

Returns the record ID of the Digital Container indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Digital Containers you must obtain a reference to the **INHRDataDigital** interface of the Data Manager. Use the **NumDigitalContainers** property to determine the number of Digital Containers available.

Example:

Get an object reference for each Digital Container configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obContainer As INHRContainer
Dim sName As String

For lCount = 0 To obDataMgrDigital.NumDigitalContainers - 1
    hID = obDataMgrDigital.GetDigitalContainerId(lCount)
    Set obContainer = obDataMgrDigital.GetDigitalContainerObj(hID)
    sName = obContainer.Describe
    Set obContainer = Nothing
Next
```

See Also:

NumDigitalContainers, GetDigitalContainerObj, Describe (Container).

4.3 GetDigitalVectorId

Returns the id for the vector at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataDigital

Function: **GetDigitalVectorId (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired vector.

Return Values:

Returns the record ID of the digital vector indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager maintains the list of all digital vectors defined in the program. To gain access to digital vectors you must obtain a reference to the INHRDataDigital interface of the Data Manager. Use the NumDigitalVectors property to determine the number of digital vectors available, and GetDigitalVectorId to obtain the record ID of each vector.

Example:

Iterate through each digital vector in the program and obtain the vector name of each digital vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrDigital.NumDigitalVectors - 1
    lVectorID = obDataMgrDigital.GetDigitalVectorId(lCount)
    sVectorName = obDataMgrDigital.GetDigitalVectorLabel(lVectorID)
Next
```

See Also:

NumDigitalVectors, GetDigitalVectorLabel.

4.4 GetDigitalVectorLabel

Gets the label for the vector id.
Member of NHRINTERFACESLib.INHRDataDigital

Function: **GetDigitalVectorLabel (**
lIDVector As Long
) As String

Parameters:

lIDVector:

The record ID of the desired digital vector.

Return Values:

Returns the contents of the Vector Name field describing the vector.

Remarks:

The Data Manager maintains the list of all digital vectors defined in the program. To gain access to digital vectors you must obtain a reference to the INHRDataDigital interface of the Data Manager. Use the NumDigitalVectors property to determine the number of digital vectors available, and GetDigitalVectorId to obtain the record ID of each vector.

The Vector Name column is located on the PowerEdit Digital pane.

Example:

Iterate through each digital vector in the program and obtain the vector name of each digital vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrDigital.NumDigitalVectors - 1
    lVectorID = obDataMgrDigital.GetDigitalVectorId(lCount)
    sVectorName = obDataMgrDigital.GetDigitalVectorLabel(lVectorID)
Next
```

See Also:

NumDigitalVectors, GetDigitalVectorId

4.5 NumDigitalContainers

The number of containers. (read-only).
Member of NHRINTERFACESLib.INHRDataDigital

Property: NumDigitalContainers As Long

Parameters:

None

Return Values:

Returns the number of Digital Containers configured in the test program.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Digital Containers you must obtain a reference to the INHRDataDigital interface of the Data Manager. Use the NumDigitalContainers property to determine the number of Digital Containers available.

Example:

Get an object reference for each Digital Container configured in the program and get its description. Be sure to release the object by setting the reference variable to Nothing when it is

```
Dim lCount As Long
Dim hID As Long
Dim obContainer As INHRContainer
Dim sName As String

For lCount = 0 To obDataMgrDigital.NumDigitalContainers - 1
    hID = obDataMgrDigital.GetDigitalContainerId(lCount)
    Set obContainer = obDataMgrDigital.GetDigitalContainerObj(hID)
    sName = obContainer.Describe
    Set obContainer = Nothing
Next
```

See Also:

GetDigitalContainerId, GetDigitalContainerObj, Describe (Container)

4.6 NumDigitalVectors

The number of vectors. (read-only).
Member of NHRINTERFACESLib.INHRDataDigital

Property: NumDigitalVectors As Long

Parameters:

None

Return Values:

Returns the number of digital vectors configured in the test program.

Remarks:

The Data Manager maintains the list of all digital vectors defined in the program. To gain access to digital vectors you must obtain a reference to the INHRDataDigital interface of the Data Manager.

Example:

Iterate through each digital vector in the program and obtain the vector name of each digital vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrDigital.NumDigitalVectors - 1
    lVectorID = obDataMgrDigital.GetDigitalVectorId(lCount)
    sVectorName = obDataMgrDigital.GetDigitalVectorLabel(lVectorID)
Next
```

See Also:

GetDigitalVectorId, GetDigitalVectorLabel

5. MEMBER OF NHRINTERFACESLIB.INHRDATAINPUT

5.1 GetDigitalContainerObj

Gets the IUnknown interface for the container associated with the container id.
Member of NHRINTERFACESLib.INHRDataInput

Function: **GetInContainerObj (**
lIDContainer As Long
) As IUnknown

Parameters:

lIDContainer:

A zero-based ID of the desired Digital Container object.

Return Values:

Returns an object reference to the Digital Container if successful, NULL if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Digital Containers you must obtain a reference to the INHRDataDigital interface of the Data Manager. Use the NumDigitalContainers property to determine the number of Digital Containers available.

Use the Set statement to assign the object reference to an appropriate variable. This may be any Interface exposed by the Digital Container object: INHRContainer, INHRDigitalContainer.

Release the object by setting the reference variable to **Nothing** when it is no longer needed.

Example:

Obtain a reference to the first Digital Container in the program and then properly release the object:

```
Dim obDigitalContainer As INHRDigitalContainer
Dim lIDContainer As Long

lIDContainer = 0
Set obDigitalContainer = obDataMgrDigital.GetDigitalContainerObj(lIdContainer)
Set obDigitalContainer = Nothing
```

See Also:

GetOutContainerObj, GetInContainerObj, NumDigitalContainers

5.2 GetInContainerId

Returns the id for the container at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataInput

Function: **GetInContainerId (**
lIndex As Long
) As Long

Parameters:

lIndex:

The zero-based index of the desired Input Container.

Return Values:

Returns the record ID of the Input Container indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Input Containers you must obtain a reference to the INHRDataInput interface of the Data Manager. Use the NumInContainers property to determine the number of Input Containers available.

Example:

Get an object reference for each Input Container configured in the program and get its description. Be sure to release the object by setting the reference variable to Nothing when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obContainer As INHRContainer
Dim sName As String

For lCount = 0 To obDataMgrIn.NumInContainers - 1
    hID = obDataMgrIn.GetInContainerId(lCount)
    Set obContainer = obDataMgrIn.GetInContainerObj(hID)
    sName = obContainer.Describe
    Set obContainer = Nothing
Next
```

See Also:

NumInContainers, GetInContainerObj, Describe (Container)

5.3 GetInContainerObj

Gets the IUnknown interface for the container associated with the container id.
Member of NHRINTERFACESLib.INHRDataInput

Function: **GetInContainerObj (**
lIDContainer As Long
) As IUnknown

Parameters:

lIDContainer:

A zero-based ID of the desired Input Container object.

Return Values:

Returns an object reference to the Input Container if successful, NULL if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Input Containers you must obtain a reference to the INHRDataInput interface of the Data Manager. Use the NumInContainers property to determine the number of Input Containers available.

Use the Set statement to assign the object reference to an appropriate variable. This may be any Interface exposed by the Input Container object: INHRContainer, INHRInputContainer, INHRDmm, INHRMeas.

Release the object by setting the reference variable to Nothing when it is no longer needed.

Example:

Obtain a reference to the first Input Container in the program and then properly release the object:

```
Dim obInContainer As INHRInputContainer
Dim lIDContainer As Long

lIDContainer = 0
Set obInContainer = obDataMgrIn.GetInContainerObj(lIdContainer)
Set obInContainer = Nothing
```

See Also:

GetOutContainerObj, GetDigitalContainerObj, NumInContainers

5.4 GetInVectorId

Returns the id for the vector at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataInput

Function: **GetInVectorId (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired vector.

Return Values:

Returns the record ID of the input vector indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager maintains the list of all input vectors defined in the program. To gain access to input vectors you must obtain a reference to the INHRDataInput interface of the Data Manager. Use the NumInVectors property to determine the number of input vectors available, and GetInVectorId to obtain the record ID of each vector.

Example:

Iterate through each input vector in the program and obtain the vector name of each input vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrIn.NumInVectors - 1
    lVectorID = obDataMgrIn.GetInVectorId(lCount)
    sVectorName = obDataMgrIn.GetInVectorLabel(lVectorID)
Next
```

See Also:

NumInVectors, GetInVectorLabel

5.5 GetInVectorLabel

Gets the label for the vector id.
Member of NHRINTERFACESLib.INHRDataInput

Function: **GetInVectorLabel (**
lIDVector As Long
) As String

Parameters:

lIDVector:

The record ID of the desired input vector.

Return Values:

Returns the contents of the Vector Name field describing the vector.

Remarks:

The Data Manager maintains the list of all input vectors defined in the program. To gain access to input vectors you must obtain a reference to the INHRDataInput interface of the Data Manager. Use the NumInVectors property to determine the number of input vectors available, and GetInVectorId to obtain the record ID of each vector.

The Vector Name column is located on the PowerEdit Input pane.

Example:

Iterate through each input vector in the program and obtain the vector name of each input vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrIn.NumInVectors - 1
    lVectorID = obDataMgrIn.GetInVectorId(lCount)
    sVectorName = obDataMgrIn.GetInVectorLabel(lVectorID)
Next
```

See Also:

NumInVectors, GetInVectorId

5.6 NumInContainers

The number of containers. (read-only).
Member of NHRINTERFACESLib.INHRDataInput

Property: NumInContainers As Long

Parameters:

None

Return Values:

Returns the number of Input Containers configured in the test program.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Input Containers you must obtain a reference to the INHRDataInput interface of the Data Manager. Use the NumInContainers property to determine the number of Input Containers available.

Example

Get an object reference for each Input Container configured in the program and get its description. Be sure to release the object by setting the reference variable to Nothing when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obContainer As INHRContainer
Dim sName As String

For lCount = 0 To obDataMgrIn.NumInContainers - 1
    hID = obDataMgrIn.GetInContainerId(lCount)
    Set obContainer = obDataMgrIn.GetInContainerObj(hID)
    sName = obContainer.Describe
    Set obContainer = Nothing
Next
```

See Also:

GetInContainerId, GetInContainerObj, Describe (Container)

5.7 NumInVectors

The number of vectors. (read-only).
Member of NHRINTERFACESLib.INHRDataInput

Property: NumInVectors As Long

Parameters:

None

Return Values:

Returns the number of input vectors configured in the test program.

Remarks:

The Data Manager maintains the list of all input vectors defined in the program. To gain access to input vectors you must obtain a reference to the INHRDataInput interface of the Data Manager.

Example:

Iterate through each input vector in the program and obtain the vector name of each input vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrIn.NumInVectors - 1
    lVectorID = obDataMgrIn.GetInVectorId(lCount)
    sVectorName = obDataMgrIn.GetInVectorLabel(lVectorID)
Next
```

See Also:

GetInVectorId, GetInVectorLabel

6. MEMBER OF NHRINTERFACESLIB.INHRDATAMEAS

6.1 GetMeasDevName

Gets the logical device name of the measurement device for the vector id.
Member of NHRINTERFACESLib.INHRDataMeas

Function: **GetMeasDevName (**
lIDVector As Long
) As String

Parameters:

lIDVector:

The record ID of the desired measurement vector.

Return Values:

Returns the logical device name of the measurement device defining the vector.

Remarks:

The Data Manager maintains the list of all measurement vectors defined in the program. To gain access to measurement vectors you must obtain a reference to the INHRDataMeas interface of the Data Manager. Use the NumMeasVectors property to determine the number of measurement vectors available, and GetMeasVectorId to obtain the record ID of each vector.

Example:

Iterate through each measurement vector in the program and obtain the logical name of the measurement device defining each vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sName As String

For lCount = 0 To obDataMgrMeas.NumMeasVectors - 1
    lVectorID = obDataMgrMeas.GetMeasVectorId(lCount)
    sName = obDataMgrMeas.GetMeasDevName(lVectorID)
Next
```

See Also:

NumMeasVectors, GetMeasVectorId

6.2 GetMeasDevProp

Gets the property blob handle for the vector id.
Member of NHRINTERFACESLib.INHRDataMeas

Function: **GetMeasDevProp (**
lIDVector As Long
) As Long

Parameters:

lIDVector:

The record ID of the desired measurement vector.

Return Values:

Returns the handle to the property blob defining the vector.

Remarks:

The Data Manager maintains the list of all measurement vectors defined in the program. To gain access to measurement vectors you must obtain a reference to the INHRDataMeas interface of the Data Manager. Use the NumMeasVectors property to determine the number of measurement vectors available, and GetMeasVectorId to obtain the record ID of each vector.

The blob handle can be passed to the measurement device using a **LoadBlobMeas** call prior to displaying the measurement user interface via **UIMeas**, or querying any specific measurement attributes such as **GetUnits**.

Example:

Iterate through each measurement vector in the program and obtain the blob handle of each measurement vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim hBlob As Long

For lCount = 0 To obDataMgrMeas.NumMeasVectors - 1
    lVectorID = obDataMgrMeas.GetMeasVectorId(lCount)
    hBlob = obDataMgrMeas.GetMeasDevProp(lVectorID)
Next
```

See Also:

NumMeasVectors, GetMeasVectorId, LoadBlobMeas, GetUnits

6.3 GetMeasVectorId

Returns the id for the vector at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataMeas

Function: **GetMeasVectorId (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired vector.

Return Values:

Returns the record ID of the measurement vector indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager maintains the list of all measurement vectors defined in the program. To gain access to measurement vectors you must obtain a reference to the INHRDataMeas interface of the Data Manager. Use the NumMeasVectors property to determine the number of measurement vectors available, and GetMeasVectorId to obtain the record ID of each vector.

Example:

Iterate through each measurement vector in the program and obtain the logical name of the measurement device defining each vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sName As String

For lCount = 0 To obDataMgrMeas.NumMeasVectors - 1
    lVectorID = obDataMgrMeas.GetMeasVectorId(lCount)
    sName = obDataMgrMeas.GetMeasDevName(lVectorID)
Next
```

See Also:

NumMeasVectors, GetMeasDevName

6.4 GetMeasVectorLabel

Gets the label for the vector id.
Member of NHRINTERFACESLib.INHRDataMeas

Function: **GetMeasVectorLabel (**
IIDVector As Long
) As String

Parameters:

IIDVector:

The record ID of the desired measurement vector.

Return Values:

Returns the contents of the Vector Name field describing the vector.

Remarks:

The Data Manager maintains the list of all measurement vectors defined in the program. To gain access to measurement vectors you must obtain a reference to the INHRDataMeas interface of the Data Manager. Use the NumMeasVectors property to determine the number of measurement vectors available, and GetMeasVectorId to obtain the record ID of each vector.

The Vector Name column is located on the PowerEdit Measure pane.

Example:

Iterate through each measurement vector in the program and obtain the vector name of each measurement vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrMeas.NumMeasVectors - 1
    lVectorID = obDataMgrMeas.GetMeasVectorId(lCount)
    sVectorName = obDataMgrMeas.GetMeasVectorLabel(lVectorID)
Next
```

See Also:

NumMeasVectors, GetMeasVectorId

6.5 NumMeasVectors

The number of vectors. (read-only).
Member of NHRINTERFACESLib.INHRDataMeas

Property: **NumMeasVectors As Long**

Parameters:

None

Return Values:

Returns the number of measurement vectors configured in the test program.

Remarks:

The Data Manager maintains the list of all measurement vectors defined in the program. To gain access to measurement vectors you must obtain a reference to the INHRDataMeas interface of the Data Manager.

Example:

Iterate through each measurement vector in the program and obtain the logical name of the measurement device defining each vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sName As String
For lCount = 0 To obDataMgrMeas.NumMeasVectors - 1
    lVectorID = obDataMgrMeas.GetMeasVectorId(lCount)
    sName = obDataMgrMeas.GetMeasDevName(lVectorID)
Next
```

See Also:

NumMeasVectors, GetMeasVectorId, GetMeasDevName

7. MEMBER OF NHRINTERFACESLIB.INHRDATAOUTPUT

7.1 GetOutContainerId

Returns the id for the container at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataOutput

Function: **GetOutContainerId (**
lIndex As Long
) As Long

Parameters:

lIndex:

The zero-based index of the desired Output Container.

Return Values:

Returns the record ID of the Output Container indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Output Containers you must obtain a reference to the INHRDataOutput interface of the Data Manager. Use the NumOutContainers property to determine the number of Output Containers available.

Example:

Get an object reference for each Output Container configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obContainer As INHRContainer
Dim sName As String

For lCount = 0 To obDataMgrOut.NumOutContainers - 1
    hID = obDataMgrOut.GetOutContainerId(lCount)
    Set obContainer = obDataMgrOut.GetOutContainerObj(hID)
    sName = obContainer.Describe
    Set obContainer = Nothing
Next
```

See Also:

NumOutContainers, GetOutContainerObj, Describe (Container)

7.2 GetOutContainerObj

Gets the IUnknown interface for the container associated with the container id.
Member of NHRINTERFACESLib.INHRDataOutput

Function: **GetOutContainerObj (**
lIDContainer As Long
) As IUnknown

Parameters:

lIDContainer:

A zero-based ID of the desired Output Container object.

Return Values:

Returns an object reference to the Output Container if successful, NULL if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Output Containers you must obtain a reference to the INHRDataOutput interface of the Data Manager. Use the NumOutContainers property to determine the number of Output Containers available.

Use the Set statement to assign the object reference to an appropriate variable. This may be any Interface exposed by the Output Container object: INHRContainer, INHROutputContainer, INHRDmm, INHRMeas.

Release the object by setting the reference variable to Nothing when it is no longer needed.

Example:

Obtain a reference to the first Output Container in the program and then properly release the object:

```
Dim obOutContainer as INHROutputContainer
Dim lIDContainer as Long

lIDContainer = 0
Set obOutContainer = obDataMgrOut.GetOutContainerObj(lIdContainer)
Set obOutContainer = Nothing
```

See Also:

GetInContainerObj, GetDigitalContainerObj, NumOutContainers

7.3 GetOutVectorId

Returns the id for the vector at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataOutput

Function: **GetOutVectorId (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired vector.

Return Values:

Returns the record ID of the output vector indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager maintains the list of all output vectors defined in the program. To gain access to output vectors you must obtain a reference to the INHRDataOutput interface of the Data Manager. Use the NumOutVectors property to determine the number of output vectors available, and GetOutVectorId to obtain the record ID of each vector.

Example:

Iterate through each output vector in the program and obtain the vector name of each output vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrOut.NumOutVectors - 1
    lVectorID = obDataMgrOut.GetOutVectorId(lCount)
    sVectorName = obDataMgrOut.GetOutVectorLabel(lVectorID)
Next
```

See Also:

NumOutVectors, GetOutVectorLabel

7.4 GetOutVectorLabel

Gets the label for the vector id.
Member of NHRINTERFACESLib.INHRDataOutput

Function: **GetOutVectorLabel (**
lIDVector As Long
) As String

Parameters:

lIDVector:

The record ID of the desired output vector.

Return Values:

Returns the contents of the Vector Name field describing the vector.

Remarks:

The Data Manager maintains the list of all output vectors defined in the program. To gain access to output vectors you must obtain a reference to the INHRDataOutput interface of the Data Manager. Use the NumOutVectors property to determine the number of output vectors available, and GetOutVectorId to obtain the record ID of each vector.

The Vector Name column is located on the PowerEdit Output pane.

Example:

Iterate through each output vector in the program and obtain the vector name of each output vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrOut.NumOutVectors - 1
    lVectorID = obDataMgrOut.GetOutVectorId(lCount)
    sVectorName = obDataMgrOut.GetOutVectorLabel(lVectorID)
Next
```

See Also:

NumOutVectors, GetOutVectorId

7.5 NumOutContainers

The number of containers. (read-only).
Member of NHRINTERFACESLib.INHRDataOutput

Property NumOutContainers As Long

Parameters:

None

Return Values:

Returns the number of Output Containers configured in the test program.

Remarks:

The Data Manager creates and controls the lifetime of all Container objects. To gain access to Output Containers you must obtain a reference to the INHRDataOutput interface of the Data Manager. Use the NumOutContainers property to determine the number of Output Containers available.

Example:

Get an object reference for each Output Container configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obContainer As INHRContainer
Dim sName As String

For lCount = 0 To obDataMgrOut.NumOutContainers - 1
    hID = obDataMgrOut.GetOutContainerId(lCount)
    Set obContainer = obDataMgrOut.GetOutContainerObj(hID)
    sName = obContainer.Describe
    Set obContainer = Nothing
Next
```

See Also:

GetOutContainerId, GetOutContainerObj, Describe (Container)

7.6 NumOutVectors

The number of vectors. (read-only).
Member of NHRINTERFACESLib.INHRDataOutput

Property NumOutVectors As Long

Parameters:

None

Return Values:

Returns the number of output vectors configured in the test program.

Remarks:

The Data Manager maintains the list of all output vectors defined in the program. To gain access to output vectors you must obtain a reference to the INHRDataOutput interface of the Data Manager.

Example:

Iterate through each output vector in the program and obtain the vector name of each output vector:

```
Dim lCount As Long
Dim lVectorID As Long
Dim sVectorName As String

For lCount = 0 To obDataMgrOut.NumOutVectors - 1
    lVectorID = obDataMgrOut.GetOutVectorId(lCount)
    sVectorName = obDataMgrOut.GetOutVectorLabel(lVectorID)
Next
```

See Also:

GetOutVectorId, GetOutVectorLabel

8. MEMBER OF NHRINTERFACESLIB.INHRDMM

8.1 Fetch

Retrieve the data and end the measurement cycle.
Member of NHRINTERFACESLib.INHRDmm

Function: **Fetch (**
lLogID As Long,
pdValue As Double,
[lNumMeasurements As Long = 1]
) As Long

Parameters:

lLogId:

The logical ID of the DMM making the measurement.

pdValue:

A double (or array of doubles) in which to place the measured value(s).

lNumMeasurements

Optional parameter (default = 1). The pdValue parameter may be passed as an array of elements in order to obtain more than one measurement using this device if the measurement was setup as an array of measurement. In this case, **lNumMeasurements** must indicate the size of the dValue array. An error will occur if the requested number of measurements exceeds the number of available values. See Setup, SetupBlobs.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

The Fetch command retrieves the measurement values obtained by the Setup (or SetupBlobs) and Meas commands. It may be repeated any number of times by itself in order to retrieve the same value from the measurement device. If a new value is desired, the combination of Meas and Fetch must be repeated.

Example:

Setup, initiate and fetch the value of a measurement defined by a measurement vector:

```
Dim dMeasurement As Double

obDMM.SetupBlobs lLogID, sMuxChannel, hBlob, dRange
obDMM.Meas lLogID
obDMM.Fetch lLogID, dMeasurement

‘ Fetch the same value again
obDMM.Fetch lLogID, dMeasurement

‘ Get a new value without changing the measurement setup
obDMM.Meas lLogID
obDMM.Fetch lLogID, dMeasurement
```

See Also:

Setup, SetupBlobs, Meas

8.2 Meas

Arm, and wait for acquisition event, if applicable.
Member of NHRINTERFACESLib.INHRDmm

Function: **Meas (**
lLogID As Long,
[dTimeout As Double],
[bSyncAll As Long = 1]
) As Long

Parameters:

lLogId:

The logical ID of the DMM making the measurement.

dTimeout:

Optional parameter (default = infinite). The maximum amount of time to wait for the measurement to finish.

bSyncAll:

Optional parameter (default = True). True to cause all tester communication to be emptied from queues before beginning the measurement cycle; False to cause the measurement to begin immediately.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

Depending on the type of measurement, this function either initiates a measurement cycle, or arms the device in anticipation of a measurement trigger.

Example:

Setup, initiate and fetch the value of a measurement defined by a measurement vector:

```
obDMM.SetupBlobs lLogID, sMuxChannel, hBlob, dRange
obDMM.Meas lLogID
obDMM.Fetch lLogID, dMeasurement
```

See Also:

Setup, SetupBlobs, Fetch

8.3 SetupBlobs

Setup the specified Dmm function using Blobs.
Member of NHRINTERFACESLib.INHRDmm

Function: **SetupBlobs (**
lLogID As Long,
sChannel As String,
phBlob As Long,
pdRange As Double,
[lNumMeasurements As Long = 1]
) As Long

Parameters:

lLogId:

The logical ID of the DMM making the measurement.

sChannel:

The logical name of the mulitplexer channel to be measured.

phBlob:

The blob handle of the measurement vector describing the measurement.

pdRange:

A maximum value that you expect will never, under normal circumstances, be exceeded by the measurement.

lNumMeasurements:

Optional parameter (default = 1). The sChannel, phBlob, and pdRange parameters may be passed as arrays of elements in order to obtain more than one measurement using this device. In this case, lNumMeasurements must indicate the size of the arrays.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

This function provides the setup information to the DMM measurement device through the use of blobs. It does not initiate a measurement.

The measurement may be made any number of times without changing the setup (see example).

Example:

Setup, then initiate and fetch the value of a measurement defined by a measurement vector five times:

```
Dim iCount As Integer

obDMM.SetupBlobs lLogID, sMuxChannel, hBlob, dRange

For i = 0 To 4
    obDMM.Meas lLogID
    obDMM.Fetch lLogID, dMeasurement
Next
```

See Also:

Setup, Meas, Fetch

9. MEMBER OF NHRINTERFACESLIB.INHRMEAS

9.1 GetUnits

Returns the units for the active measurement.
Member of NHRINTERFACESLib.INHRMeas

Function: **GetUnits (**
lLogID As Long
) As String

Parameters:

lLogID:

The logical ID of the measurement device.

Return Values:

Returns the units string for the currently loaded measurement vector.

Remarks:

Before using this function you must load a measurement vector (stored as a blob) into the measurement device. Once loaded, you may edit it visually by calling UIMeas, or query various attributes of the vector such as GetUnits or GetDescribe.

To gain access to this function you must obtain a reference to the INHRMeas interface of the desired measurement device.

Example:

Get the units of the measurement referenced by hBlob from the selected device:

```
Dim obMeas As INHRMeas
Dim lLogID As Long

Set obMeas = g_obCfgMgr.GetLogicalDev(sDevName, lLogID)
obMeas.LoadBlobMeas hBlob, lLogID
sUnits = obMeas.GetUnits(lLogID)
Set obMeas = Nothing
```

See Also:

GetLogicalDev, LoadBlobMeas, GetMeasDevProp

9.2 LoadBlobMeas

Load the measurement property data referred to by hBlob from the Blob Manager.
Member of NHRINTERFACESLib.INHRMeas

Function: **LoadBlobMeas (**
hBlob As Long,
lLogID As Long
) As Long

Parameters:

hBlob:

The blob handle referencing the desired measurement.

lLogID:

The logical ID of the measurement device.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

Use this function to load a measurement vector (stored as a blob) into a measurement device. Once loaded, you may edit it visually by calling UIMeas, or query various attributes of the vector such as GetUnits or GetDescribe.

To gain access to this function you must obtain a reference to the INHRMeas interface of the desired measurement device.

Example:

Get the units of the measurement referenced by hBlob from the selected device:

```
Dim obMeas As INHRMeas
Dim lLogID As Long

Set obMeas = g_obCfgMgr.GetLogicalDev(sDevName, lLogID)
obMeas.LoadBlobMeas hBlob, lLogID
sUnits = obMeas.GetUnits(lLogID)
Set obMeas = Nothing
```

See Also:

GetLogicalDev, GetUnits, GetMeasDevProp

10. MEMBER OF NHRINTERFACESLIB.INHRDEV

10.1 GetLogicalDevSubType

The sub-type of the logical device. (Read-only)
Member of NHRINTERFACESLib.INHRDev

Function: **GetLogicalDevSubType (**
lLogID As Long
) As String

Parameters:

lLogID:

The logical ID of the device.

Return Values:

Returns the subtype descriptor of the device.

Remarks:

Sub-types are devices specific, and designed to indicate a variation of a more generic device type. While AC and DC sources are both "source" devices, one has an "AC" subtype while the other has a "DC" subtype. In the example below, the measurement device subtype is "DMM". In general, the subtype of a device can be determined by examining the first part of the automatically generated logical name:

- "AC Source 1" – subtype is "AC"
- "DMM Measurement 2" – subtype is "DMM"
- "DIN Measurement 1" – subtype is "DIN"
- "External Mux 1" – subtype is "External"

Use this function to programmatically query the subtype from the device directly.

Example:

Obtain a reference to the device named "DMM Measurement 1" in the program, get its sub-type and then properly release the object:

```
Dim obDev As INHRDev
Dim lLogId As Long
Dim sSubType As String

Set obDev = g_obCfgMgr.GetLogicalDev("DMM Measurement 1", lLogId)
sSubType = obDev.GetLogicalDevSubType(lLogId)
MsgBox "The sub-type is " & sSubType, vbInformation
Set obDev = Nothing
```

See Also:

GetLogicalDev

11. MEMBER OF NHRINTERFACESLIB.INHRTR

11.1 CleanUp

Unused function. Never called by emPower clients
Member of NHRINTERFACESLib.INHRTR

Sub CleanUp (
[sLogicalDevName As String]
)

Parameters:

sLogicalDevName:

Reserved parameter

Return Values:

None

Remarks:

This is an unused function. It is never called by emPower clients

Implementation Example:

Do not put any code in the body of this subroutine.

See Also:

None

11.2 CopyBlobProp

Copies the property data into a new storage location in the blob manager. Loads the hBlob before the copy is performed.
Member of NHRINTERFACESLib.INHRTR

Function: **CopyBlobProp (**
hBlob As Long,
[lLogID As Long]
) As Long

Parameters:

hBlob:

The handle of the blob to copy.

lLogID:

Optional parameter (default = 0). The logical ID of the test routine. Currently always set to zero by emPower clients.

Return Values:

Returns the hBlob of the new storage location if successful, 0 if failed.

Remarks:

This function is used to make a copy of the desired property blob. Load the property structure referenced by hBlob into the test routine object prior to saving to a new location. By specifying the new blob location to be zero, the Blob Manager will create a new entry and return the handle value to the new location.

If the property blob contains handles to other blobs (e.g., an SPC property blob or a UUT Communications blob) you must copy those blob first by calling the CopyBlobProp function of the appropriate component. Failure to do so will cause the two property blobs to reference the same embedded property blob. This cross-coupling will have the effect of the edits in one object showing up in the other object instead of the independence, which would be expected with a copy.

Implementation Example 1:

Make a copy of the provided property blob:

```
If INHRTR_LoadBlobProp(hBlob, lLogID) Then
    INHRTR_CopyBlobProp = INHRTR_SaveBlobProp(0, lLogID)
Else
    INHRTR_CopyBlobProp = 0
End If
```

Implementation Example 2:

Copying a property blob that contains handles to other property blobs:

```
If INHRTR_LoadBlobProp(hBlob, lLogID) Then
    ' Copy any SPC and/or Communications blobs first
    If g_blobProp.hSPC <> 0 Then
        g_blobProp.hSPC = g_obSPCSvr.CopyBlobProp(g_blobProp.hSPC)
    End If
    INHRTR_CopyBlobProp = INHRTR_SaveBlobProp(0, lLogID)
Else
    INHRTR_CopyBlobProp = 0
End If
```

See Also:

LoadBlobProp, SaveBlobProp

11.3 DeleteBlobProp

Delete the property data referred to by hBlob from the blob manager.
Member of NHRINTERFACESLib.INHRTR

Function: **DeleteBlobProp (**
hBlob As Long,
[lLogID As Long]
) As Long

Parameters:

hBlob:

The handle of the blob to delete.

lLogID>:

Optional parameter (default = 0). The logical ID of the test routine. Currently always set to zero by emPower clients.

Return Values:

Return True if the blob (and any embedded property blobs) were successfully deleted, otherwise False.

Remarks:

This function is used to delete the desired property blob. If the property blob contains handles to other blobs (e.g., an SPC property blob or a UUT Communications blob) you must delete those blobs first by calling the DeleteBlobProp function of the appropriate component. Failure to do so will leave those property blobs in the Blob Manager memory with no way to access them. This is referred to as an "orphaned" blob.

Implementation Example 1:

Delete the provided property blob:

```
g_obDataMgr.BlobMgr.DeleteBlob hBlob
INHRTR_DeleteBlobProp = True
```

Implementation Example 2:

Deleting a property blob that contains handles to other property blobs:

```
' Delete any SPC and/or Communications blobs first
If g_blobProp.hSPC <> 0 Then
    g_obSPCSvr.DeleteBlobProp g_blobProp.hSPC
End If

g_obDataMgr.BlobMgr.DeleteBlob hBlob
INHRTR_DeleteBlobProp = True
```

See Also:

LoadBlobProp, SaveBlobProp

11.4 Describe (Test Routine)

Description for the test routine. (Read-only)
Member of NHRINTERFACESLib.INHRTR

Property Describe As String

Parameters:

None

Return Values:

Return the description of the test routine.

Remarks:

The description is encoded with information defining the category tree in which to place this entry in the test routine dialog. The syntax for this encryption is: name[;|,]category]... where (;) = New root, (,) = child

The test routine entry is inserted into the tree structure as specified; if any category (root or child) does not exist in the test routine tree structure it will be created.

For example, Multi-Pass Regulation; Custom, Power Supply; Custom, UPS would create the following entries:

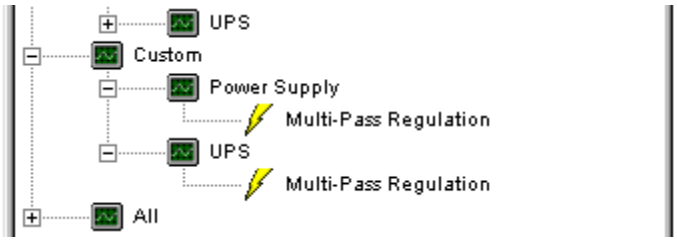

Figure 1 – The Custom Test Routine node (expanded)

Implementation Example:

Typical Describe procedure:

```
INHRTR_Describe = sNHRTRName
```

where sNHRTRName is defined as:

```
Public Const sNHRTRName = "Multi-Pass Regulation;Custom,Power Supply;Custom,UPS"
```

See Also:

LoadBlobProp, SaveBlobProp

11.5 DumpProp

A human readable dump of the properties.
Member of NHRINTERFACESLib.INHRTR

Function: **DumpProp (**
[bPrinterFormat As Long]
[lLogID As Long]
) As String

Parameters:

bPrinterFormat:

Optional parameter (default = -1). The type of dump and/or the initial indent level of the dump.

lLogID:

Optional parameter (default = 0). The logical ID of the test routine. Currently always set to zero by emPower clients.

Return Values:

Return the properties in a human readable string.

Remarks:

This function should support two different types of printing formats: a horizontally favored summary print, and a vertically favored full print. The client choice is encoded in the **bPrinterFormat** parameter. If **bPrinterFormat** equals –2, the client is requesting a summary print; any other value is a request for a full print where **bPrinterFormat** indicates the number of indentation levels the dump should begin at.

Implementation Example:

See the Test Routine Template code for an example of a simple DumpProp function.

See Also:

LoadBlobProp, SaveBlobProp

11.6 Go

Performs the test. Returns the number of errors
Member of NHRINTERFACESLib.INHRTR

Function: **Go (**
bDebug As Long,
InitialVectorID As Long,
[bAlwaysFail As Long]
) As Long

Parameters:

bedbug:

If True, the routine should log any additional information deemed necessary for debugging.

InitialVectorID:

Reserved parameter. Currently always set to zero by emPower clients.

bAlwaysFail:

Optional parameter (default = False). If True, the routine should log all test results as failures.

Return Values:

Return the total number of failing measurements and errors encountered during test execution. Returning zero indicates a Pass result, non-zero indicates a Fail result.

Remarks:

This function is called by the application to execute the test routine algorithm using the parameters currently loaded in the property blob.

It is extremely important that the test routine returns the proper error count at the completion of the algorithm. If a failing measurement is logged by the test but the return value is set to zero, calling applications will assume the test passed. In addition, any faults logged by the routine should increment the returned failure count.

If **bDebug** is True, the routine is being instructed to provide any debug information built into the algorithm. This may consist of loop count values, intermediate results or any other information that may help the user debug their use of this routine without having to single-step through the code in the debugger. For example, a looping routine may log both the set and measured values for each loop.

bAlwaysFail is used to test the sequencing of the test program. When True, the test routine should log every measurement as a failure. This is easily accomplished if the routine is using the **EvaluateValResult** or **EvaluateStrResult** functions (provided by the Helper object) when validating results as these functions have a parameter explicitly for this feature.

Implementation Example:

See the Test Routine Template code for an example of a simple Go function.

See Also:

EvaluateStrResult, EvaluateValResult, LoadBlobProp, SaveBlobProp

11.7 Init (Test Routine)

Initializes the test routine.
Member of NHRINTERFACESLib.INHRTR

Sub Init (
hwndParent As Long,
pdispDatalogServer As Object,
pdispDataManager As Object,
pdispCfgManager As Object,
pdispHelperSvr As Object,
dispFaultMgr As Object,
pdispTestInfoSvr As Object
)

Parameters:

hwndParent:

The window handle of the calling application.

pdispDatalogServer:

The IDispatch interface of the active Datalog Server object.

pdispDataManager:

The IDispatch interface of the active Data Manager object.

pdispCfgManager:

The IDispatch interface of the active Configuration Server object.

pdispHelperSvr:

The IDispatch interface of the active Helper Server object.

pdispFaultMgr:

The IDispatch interface of the active Fault Manager object.

pdispTestInfoSvr:

The IDispatch interface of the active Test Information object.

Return Values:

None

Remarks:

This function is called once by the creating application. Each of the object references passed in through the parameter list is a pointer to the IDispatch interface of the associated object. You may re-assign these references to those of your choosing by using the Set statement.

Implementation Example:

Typical initialization procedure:

```
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
Set g_obSPCSvr = Nothing
```

See Also:

LoadBlobProp, SaveBlobProp

11.8 LoadBlobProp

Load the property data referred to by hBlob from the blob manager.
Member of NHRINTERFACESLib.INHRTR

Function: **LoadBlobProp (**
hBlob As Long,
[lLogID As Long]
) As Long

Parameters:

hBlob:

The handle of the blob to load.

lLogID:

Optional parameter (default = 0). The logical ID of the test routine. Currently always set to zero by emPower clients.

Return Values:

Return True if the blob was loaded successfully (and converted, if necessary), otherwise False.

Remarks:

This function is used to load the desired property blob into the property structure. This may be in preparation of running the test algorithm (Go), editing the parameters (UIProp), printing the parameters (DumpProp), etc.

It is in this function that any version conversions will take place. If the stored blob schema is less than the current schema, the stored blob must be upgraded to the current version; no conversion is necessary if the schemas are equal. An error should be reported if the stored blob schema is greater than the current schema.

Implementation Example:

See the Test Routine Template code for an example of a simple LoadBlobProp function.

See Also:

DeleteBlobProp, DumpProp, Go, SaveBlobProp, UIProp

11.9 SaveBlobProp

Saves the property data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created.
Member of NHRINTERFACESLib.INHRTR

Function: **SaveBlobProp (**
hBlob As Long,
[lLogID As Long]
) As Long

Parameters:

hBlob:

The location within the Blob Manager to store the blob.

lLogID:

Optional parameter (default = 0). The logical ID of the test routine. Currently always set to zero by emPower clients.

Return Values:

Return the hBlob if successful, 0 if failed.

Remarks:

This function is used to save the property blob in the Blob Manager. The caller is responsible for determining where the property blob should be stored.

Implementation Example:

See the Test Routine Template code for an example of a simple SaveBlobProp function.

See Also:

DeleteBlobProp, LoadBlobProp, StoreBlob

11.10 SPC (Quality Monitor)

Quality Monitor in use flag. (Read-only)
Member of NHRINTERFACESLib.INHRTR

Property SPC As Long

Parameters:

None

Return Values:

Return **True** if any Quality Monitor data collection is being performed by this routine, otherwise **False**.

Remarks:

This function is called by the application to determine if any Quality Monitor(QM) data will be logged by the routine when it is run. If the property blob contains at least one non-zero handle to an QM property blob it should return True.

Implementation Example:

Typical QM procedure:

```
If g_blobProp.hQM <> 0 Then
    INHRTR_SPC = True
Else
    INHRTR_SPC = False
End If
```

See Also:

LoadBlobProp, SaveBlobProp

11.11 UIProp

Display the properties user interface.
Member of NHRINTERFACESLib.INHRTR

Function: UIProp () As Long

Parameters:

None

Return Values:

Return **True** if the user pressed OK, **False** if the user pressed Cancel.

Remarks:

This function is called by the application to view the test routine properties using the parameters currently loaded in the property blob. The user is allowed to modify the values.

Note: do **not** automatically store the property changes in the Blob Manager should the user press "OK". The client application is responsible for determining if and where the property blob should be stored.

Implementation Example:

See the Test Routine Template code for an example of a simple UIProp function.

See Also:

LoadBlobProp, SaveBlobProp

12. MEMBER OF NHRINTERFACESLIB.INHRTESTINFO

12.1 DeleteBlobCfg

Delete the configuration data referred to by hBlob from the blob manager.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **DeleteBlobCfg (**
hBlob As Long
) As Long

Parameters:

hBlob:

The handle of the blob to delete.

Return Values:

Return True if the blob was successfully deleted, otherwise False.

Remarks:

This function is used to delete the desired configuration blob.

Implementation Example:

Delete the provided configuration blob:

```
g_obBlobMgr.DeleteBlob hBlob
INHRTestInfo_DeleteBlobCfg = True
```

See Also:

LoadBlobCfg, SaveBlobCfg

12.2 DumpCfg

A human readable dump of the configuration.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **DeleteBlobCfg (**
[lIndentLevel As Long]
) As String

Parameters:

lIndentLevel:

The initial indent level of the dump.

Return Values:

Return the configuration properties in a human readable string.

Remarks:

This function is used to create a human readable version of the configuration blob. The initial indentation level should be set to lIndentLevel. The data should be complete enough to enable a programmer to recreate the configuration setup from the printout.

Implementation Example:

See the Test Information sample code for an example of a simple DumpCfg function.

See Also:

LoadBlobCfg, SaveBlobCfg

12.3 GetFieldInfo

lIndex is the 0 based index of the field to retrieve.
Member of NHRINTERFACESLib.INHRTestInfo

Sub GetFieldInfo (
lIndex As Long,
psPrompt As String,
psData As String
)

Parameters;

lIndex:

The zero-based index of the Test Information field to retrieve.

psPrompt:

A string variable that will be loaded with the contents prompt string.

psData:

A string variable that will be loaded with the data entered by the operator in response to the prompt.

Return Values:

None

Remarks:

This function is used to retrieve the contents of the test information. Use GetNumFields to determine the number of data entry fields presented by the Test Information prompt.

Implementation Example:

Retrieve the test information data at a user-defined index:

```
If lIndex < 0 Or lIndex >= g_blobCfgHdr.lNumUserFields Then
    Exit Sub
End If

psPrompt = frmTestInfoEntry.lblPrompt(lIndex + 1).Caption
psData = frmTestInfoEntry.txtData(lIndex + 1).Text
```

See Also:

GetNumFields

12.4 GetNumFields

Returns the number of entry fields.
Member of NHRINTERFACESLib.INHRTestInfo

Function: GetNumFields () As Long

Parameters:

None

Return Values:

Return the number of prompt/data pairs.

Remarks:

This function is used to determine the number of data entry fields presented by the Test Information prompt.

Implementation Example:

Return the number of test information data sets:

INHRTestInfo_GetNumFields = g_blobCfgHdr.lNumUserFields

See Also:

GetFieldInfo

12.5 GetTestInfo

lIndex is the 0 based index of the field to retrieve.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **GetTestInfo (**
psTestInfo As String
) As Long

Parameters:

psTestInfo:

A string variable that will be loaded with the every prompt/data pair.

Return Values:

Return the value of the active fixture selected by the operator.

Remarks:

GetTestInfo will begin by displaying the user interface in order to collect the requested data from the operator. Programmers may suppress the UI by pre-filling **psTestInfo** with "NoUI". In this way, the contents of the test information may be retrieved at a later time without redisplaying the UI.

Fill **psTestInfo** with a tab-delimited list of the prompt and user data fields. Every prompt/data pair is tab-delimited and concatenated together into this single string. Note: the previous contents of the string should be overwritten (erasing the "NoUI" if necessary).

The return value depends on the active fixture selected by the user:
0 = Fixture A
1 = Fixture B
-1 = Cancel

Implementation Example:

Retrieve the test information data at a user-defined index:

```
'If sTestInfo = "NoUI" then NO UI IS DISPLAYED
'sTestInfo is set to info string and the current fixture index is returned (-1 if
none).
Function INHRTestInfo_GetTestInfo(sTestInfo As String) As Long
    If StrComp(sTestInfo, "NoUI", vbTextCompare) <> 0 Then
        frmTestInfoEntry.ShowTestInfoDialog
    End If

    If g_lFixtIdx >= 0 Then
        sTestInfo = g_sTestInfo      ' g_sTestInfo contains the tab delimited
data pairs
        INHRTestInfo_GetTestInfo = g_lFixtIdx       ' g_lFixtIdx is the Active
fixture
    Else
        INHRTestInfo_GetTestInfo = -1
    End If
End Function
```

See Also:

GetNumFields

12.6 Init (Test Information)

Initializes the test information control.
Member of NHRINTERFACESLib.INHRTestInfo

Sub Init (
hwndParent As Long,
pobCfgMgr As Object,
pobDataMgr As Object,
obFaultMgr As Object,
obHelperSvr As Object,
bOnline As Long
)

Parameters:

hwndParent:

The window handle of the calling application.

pobCfgMgr:

The IDispatch interface of the active Configuration Server object.

pobDataMgr:

The IDispatch interface of the active Data Manager object.

pobFaultMgr:

The IDispatch interface of the active Fault Manager object.

pobHelperSvr:

The IDispatch interface of the active Helper Server object.

bOnline:

True if the system is running online; **False** if running offline.

Return Values:

None

Remarks:

This function is called once by the creating application. Each of the object references passed in through the parameter list is a pointer to the IDispatch interface of the associated object. You may re-assign these references to those of your choosing by using the Set statement.

Implementation Example:

Typical initialization procedure:

```
Sub INHRTestInfo_Init(ByVal hwndParent As Long, _
        ByVal obCfgMgr As Object, _
        ByVal obDataMgr As Object, _
        ByVal obFaultMgr As Object, _
        ByVal obHelperSvr As Object, _
        ByVal bOnline As BOOL)

    Set g_obBlobMgr = obDataMgr.BlobMgr
    Set g_obHelperSvr = obHelperSvr
    Set g_obFaultMgr = obFaultMgr
    Set g_obDataMgr = obDataMgr
    Set g_obCfgMgr = obCfgMgr
    g_hwndParent = hwndParent
    g_bOnline = bOnline
End Sub
```

See Also:

None

12.7 LoadBlobCfg

Load the configuration data referred to by hBlob from the blob manager.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **LoadBlobProp (**
hBlob As Long,
) As Long

Parameters:

hBlob:

The handle of the blob to load.

Return Values:

Return True if the blob was loaded successfully (and converted, if necessary), otherwise False.

Remarks:

This function is used to load the desired configuration blob into the property structure.

It is in this function that any version conversions will take place. If the stored blob schema is less than the current schema, the stored blob must be upgraded to the current version; no conversion is necessary if the schemas are equal. An error should be reported if the stored blob schema is greater than the current schema.

Implementation Example:

See the Test Information sample code for an example of a simple LoadBlobCfg function.

See Also:

DeleteBlobCfg, Dumpcfg, SaveBlobCfg

12.8 LoadDialog

Call this to pre-load the testinfo prompt and retrieve the window handle.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **LoadDialog (**
hWndCfg As Long,
hWndEntry As Long
) As Long

Parameters:

hWndCfg:

Receives the window handle to the configuration form

hWndEntry:

Receives the window handle to the data entry form

Return Values:

Return True if the forms were loaded successfully, otherwise False.

Remarks:

This function will load the configuration and data entry forms into memory without visually displaying them. The window handles are copied into the provided variables.

Currently not called by any emPower client.

Implementation Example:

Load both the configuration and data entry forms into memory without displaying them:

```
Private Function INHRTestInfo_LoadDialog(hWndCfg As Long, hWndEntry As Long) As
Long
    Load frmTestInfoCfg
    Load frmTestInfoEntry
    hWndCfg = frmTestInfoCfg.hWnd
    hWndEntry = frmTestInfoEntry.hWnd
    INHRTestInfo_LoadDialog = True
End Function
```

See Also:

None

12.9 SaveBlobCfg

Saves the configuration data to the blob manager at the specified hBlob. If hBlob is 0, a new storage location is created.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **SaveBlobCfg (**
hBlob As Long,
) As Long

Parameters:

hBlob:

The location within the Blob Manager to store the blob.

Return Values:

Return the hBlob if successful, 0 if failed.

Remarks:

This function is used to save the configuration blob in the Blob Manager. The caller is responsible for determining where the property blob should be stored.

Implementation Example:

See the Test Information sample code for an example of a simple SaveBlobCfg function.

See Also:

DeleteBlobCfg, LoadBlobCfg, StoreBlob

12.10 SetZPos

Call this to routine to control the placement of the test info prompt in the windows Z order.
Member of NHRINTERFACESLib.INHRTestInfo

Sub SetZPos (
lInsertAfter As Long
)

Parameters:

lInsertAfter:

The test info prompt should be placed after the window referenced by this window handle

Return Values:

None

Remarks:

This function will move the data entry window in respect to the window Z order. Most windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.

lInsertAfter must be a window handle or one of the following values:
HWND_BOTTOM - Places the window at the bottom of the Z order
HWND_NOTOPMOST - Places the window above all non-topmost windows
HWND_TOP - Places the window at the top of the Z order
HWND_TOPMOST - Places the window above all non-topmost windows

Currently not called by any emPower client.

Implementation Example:

Place the data entry window at the desired Z order without changing its placement or size:

```
Sub INHRTestInfo_SetZPos(ByVal lInsertAfter As Long)
    If frmTestInfoEntry.Visible Then
        Dim lFlags As Long

        lFlags = SWP_NOMOVE Or SWP_NOSIZE
        SetWindowPos frmTestInfoEntry.hWnd, lInsertAfter, 0, 0, 0, 0, lFlags
    End If
End Sub
```

See Also:

SetWindowPos (Platform API)

12.11 UICfg

Display the configuration user interface.
Member of NHRINTERFACESLib.INHRTestInfo

Function: **UICfg () As Long**

Parameters:

None

Return Values:

Return **True** if the user pressed OK, **False** if the user pressed Cancel.

Remarks:

This function is called by the application to view the Test Information configuration using the parameters currently loaded in the configuration blob. The user is allowed to modify the values.

Note: do **not** automatically store the configuration changes in the Blob Manager should the user press "OK". The client application is responsible for determining if and where the configuration blob should be stored.

Implementation Example:

See the Test Information sample code for an example of a simple UICfg function.

See Also:

LoadBlobCfg, SaveBlobCfg

13. MEMBER OF NHRINTERFACESLIB.INHRUUTCOMM

13.1 Describe (UUT Communications)

Sets or gets a user defined description.
Member of NHRINTERFACESLib.INHRUUTComm, NHRINTERFACESLib. INHRUUTComm2

Property: Describe As String

Parameters:

Sets the user defined description of the UUT Communications object

Return Values:

Returns the user defined description of the UUT Communications object

Remarks:

The Data Manager creates and controls the lifetime of all UUT Communications objects. To gain access to UUT Communications objects you must obtain a reference to the **INHRDataUUTComm** interface of the Data Manager. Use the **NumUUTCommObjs** property to determine the number of UUT Communications objects available.

Example:

Get an object reference for each UUT Communications object configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obComm As INHRUUTComm
Dim sName As String

For lCount = 0 To obDataUUTComm.NumUUTCommObjs - 1
    hID = obDataUUTComm.GetUUTCommObjId(lCount)
    Set obComm = obDataUUTComm.GetUUTCommObj(hID)
    sName = obComm.Describe
    Set obComm = Nothing
Next
```

See Also:

NumUUTCommObjs, GetUUTCommObjId, GetUUTCommObj

13.2 GetUUTCommObj

Gets the IUnknown interface for the driver associated with the driver id.
Member of NHRINTERFACESLib.INHRDataUUTComm

Function: **GetUUTCommObj (**
lIdDriver As Long
) As IUnknown

Parameters:

lIdDriver:

A zero-based ID of the desired UUT Communications object.

Return Values:

Returns an object reference to the UUT Communications object if successful, NULL if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all UUT Communications objects. To gain access to UUT Communications objects you must obtain a reference to the **INHRDataUUTComm** interface of the Data Manager. Use the **NumUUTCommObjs** property to determine the number of UUT Communications objects available.

Use the **Set** statement to assign the object reference to an appropriate variable. This may be any Interface exposed by the UUT Communications object. All UUT Communications objects support the **INHRUUTComm** and **INHRUUTComm2** interfaces, in addition to their bus specific interface (Serial, GPIB, CAN, I^2C).

Release the object by setting the reference variable to **Nothing** when it is no longer needed.

Example:

Get an object reference for each UUT Communications object configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obComm As INHRUUTComm
Dim sName As String

For lCount = 0 To obDataUUTComm.NumUUTCommObjs - 1
    hID = obDataUUTComm.GetUUTCommObjId(lCount)
    Set obComm = obDataUUTComm.GetUUTCommObj(hID)
    sName = obComm.Describe
    Set obComm = Nothing
Next
```

See Also:

NumUUTCommObjs, GetUUTCommObjId

13.3 GetUUTCommObjId

Returns the id for the driver at lIndex (0 based). Returns 0 on error.
Member of NHRINTERFACESLib.INHRDataUUTComm

Function: **GetUUTCommObjId (**
lIndex As Long
) As Long

Parameters:

lIndex:

The zero-based index of the desired UUT Communications object.

Return Values:

Returns the record ID of the UUT Communications object indicated by the lIndex parameter, or zero if an error occurred.

Remarks:

The Data Manager creates and controls the lifetime of all UUT Communications objects. To gain access to UUT Communications objects you must obtain a reference to the **INHRDataUUTComm** interface of the Data Manager. Use the **NumUUTCommObjs** property to determine the number of UUT Communications objects available.

Example:

Get an object reference for each UUT Communications object configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obComm As INHRUUTComm
Dim sName As String

For lCount = 0 To obDataUUTComm.NumUUTCommObjs - 1
    hID = obDataUUTComm.GetUUTCommObjId(lCount)
    Set obComm = obDataUUTComm.GetUUTCommObj(hID)
    sName = obComm.Describe
    Set obComm = Nothing
Next
```

See Also:

NumUUTCommObjs, GetUUTCommObj, Describe (UUT Communications)

13.4 NumUUTCommObjs

The number of drivers. (read-only).
Member of NHRINTERFACESLib.INHRDataUUTComm

Property NumUUTCommObjs As Long

Parameters;

None

Return Values:

Returns the number of UUT Communications objects configured in the test program.

Remarks:

The Data Manager creates and controls the lifetime of all UUT Communications objects. To gain access to UUT Communications objects you must obtain a reference to the **INHRDataUUTComm** interface of the Data Manager. Use the **NumUUTCommObjs** property to determine the number of UUT Communications objects available.

Example:

Get an object reference for each UUT Communications object configured in the program and get its description. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim hID As Long
Dim obComm As INHRUUTComm
Dim sName As String

For lCount = 0 To obDataUUTComm.NumUUTCommObjs - 1
    hID = obDataUUTComm.GetUUTCommObjId(lCount)
    Set obComm = obDataUUTComm.GetUUTCommObj(hID)
    sName = obComm.Describe
    Set obComm = Nothing
Next
```

See Also:

GetUUTCommObjId, GetUUTCommObj, Describe (UUT Communications)

14. MEMBER OF NHRINTERFACESLIB.NHRBLOB

14.1 BlobLen

Returns the length of a blob.
Member of NHRINTERFACESLib.NHRBlob

Function: **BlobLen (**
lhBlob As Long
) As Long

Parameters:

lhBlob:

The handle to desired blob.

Return Values:

The length (in bytes) of the indicated blob.

Remarks:

Use this function to determine the size (in bytes) of the blob. You need this information in order to allocate enough memory to hold the contents of the blob before loading it from the Blob Manager. See the example in **RetrieveBlob**.

Example:

Get the length of a blob:

```
Dim lBlobSize As Long

lBlobSize = obBlobMgr.BlobLen(hBlob)
MsgBox "The blob length is " & lBlobSize, vbInformation
```

See Also:

BlobMgr, BlobStruct, Schema, RetrieveBlob

14.2 BlobStruct

Returns a string describing the data structure for this blob.
Member of NHRINTERFACESLib.NHRBlob

Function: **BlobStruct (**
lhBlob As Long
) As String

Parameters:

lhBlob:

The handle to desired blob.

Return Values:

A string containing the description of the data structure as provided by the blob creator.

Remarks:

The blob structure description is used to determine the type of structure stored in the blob. In general, an object should not attempt to load a blob that it doesn't know how to create. If an object supports this type of structure, it may continue to load the contents of the blob into memory.

Example:

Verify that the blob structure referenced by hBlob is a "My Structure Type":

```
'Verify data struct type
If RTrim$(obBlobMgr.BlobStruct(hBlob)) <> "My Structure Type" Then
    MsgBox "Incompatible property data!", vbCritical
End If
```

See Also:

BlobMgr, Schema, BlobLen

14.3 RetrieveBlob

Retrieves a blob from the Blob Manager's memory.
Member of NHRINTERFACESLib.NHRBlob

Function: **RetrieveBlob (**
lhBlob As Long
pvntBlob as Variant
) As Long

Parameters:

lhBlob:

The handle to desired blob.

pvntBlob:

A Variant in which to load the blob.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

This function will retrieve the desired blob from the Blob Manager into the variant parameter.

Example:

Retrieve a blob from the Blob Manager and copy it into our property structure:

```
Dim lBlobSize As Long
Dim aBytes() As Byte
Dim var As Variant
Dim blobProp As MyPropertyStructure

' Get the data from the blob manager into a variant
lBlobSize = obBlobMgr.BlobLen(hBlob)
ReDim aBytes(lBlobSize - 1)
var = aBytes
If obBlobMgr.RetrieveBlob(hBlob, var) = 0 Then
    MsgBox "Unable to retrieve property data", vbCritical
Else
    'Load properties from variant
    aBytes = var
    CopyMemory blobProp, aBytes(LBound(aBytes)), LenB(blobProp)
End If
```

See Also:

BlobMgr, BlobStruct, Schema, BlobLen, StoreBlob

14.4 Schema

Returns the schema of a blob.
Member of NHRINTERFACESLib.NHRBlob

Function: **Schema (**
lhBlob As Long
) As Long

Parameters:

lhBlob:

The handle to desired blob.

Return Values:

The schema (or version) of the indicated blob.

Remarks:

Use this value to determine the schema (or version) of the blob referenced by the handle. If the schema is earlier than the current version expected by the object you might convert the blob to the latest schema to remain backward compatible with older programs; if the schema is newer than what the object supports you should error.

Example:

Verify that the version of the blob referenced by hBlob is not greater than 1:

```
'Verify blob schema
If obBlobMgr.Schema(hBlob)) > 1 Then
    MsgBox "Incompatible property data!", vbCritical
End If
```

See Also:

BlobMgr, BlobStruct, BlobLen

14.5 StoreBlob

Stores the blob in the Blob Manager's memory.
Member of NHRINTERFACESLib.NHRBlob

```
Sub StoreBlob (
                plhBlob As Long,
                sCreatorProgId As String,
                sBlobStruct As String,
                lSchema As Long,
                pvntBlob As Variant,
                lLen As Long
              )
```

Parameters:

plhBlob:

The handle within the Blob Manager in which to store the blob. This value may be modified by the call (see Remarks).

sCreatorProgID:

A string describing the creator of the blob. This is usually the ProgID (programmatic identifier, a human-readable version of the class identifier used to identify COM/ActiveX objects) of the creating object. For emPower test routine programming, this is usually the sCreatorProgID constant found in the main .BAS module.

sBlobStruct:

A string describing the structure of the blob.

lSchema:

The schema (or version) of the blob.

pvntBlob:

A variant from which to store the blob.

lLen:

The length of the blob, in bytes.

Return Values:

None. The Blob Manager will raise an E_FAIL exception if the operation fails.

Remarks:

This function is used to store blob structures in the Blob Manager. In normal operation, the first time you store a blob you should specify a blob handle of zero in the **plhBlob** parameter. When this occurs, the Blob Manager will find a location for your data and return the handle to that location by overwriting the value of the **plhBlob** parameter. Subsequent modifications of that blob data should be stored overtop of the original data by supplying the modified **plhBlob** parameter instead of zero. Do not pass a numeric value for **plhBlob,** as the function will modify this parameter if the value is zero.

Example:

Store our property structure in the Blob Manger. Let the Blob Manager determine a new location for our blob by specifying a blob handle of zero. After the call, the handle should be non-zero:

```
Dim blobProp As MyPropertyStructure
Dim lBlobSize As Long
Dim aBytes() As Byte
Dim var As Variant
Dim hBlob As Long
Dim sBlobStruct As String

lBlobSize = LenB(blobProp)
ReDim aBytes(lBlobSize - 1) As Byte

CopyMemory aBytes(LBound(aBytes)), blobProp, lBlobSize
```

```
var = aBytes

‘ Store the blob in a new location
hBlob = 0
obBlobMgr.StoreBlob hBlob, sCreatorProgID, “MyPropertyStructure”, 1, var,
lBlobSize

‘ hBlob should not be zero any longer
If hBlob = 0 Then
    MsgBox "Unable to store blob in Blob Manager", vbCritical
End If
```

See Also:

BlobMgr, BlobStruct, Schema, BlobLen, RetrieveBlob

15. MEMBER OF NHRINTERFACESLIB.NHRCFG

15.1 GetLogicalDev

Get pointer to object and logical ID from logical name.
Member of NHRINTERFACESLib.NHRCfg, NHRINTERFACESLib.INHRCfg2, NHRINTERFACESLib.INHRCfg3

Function: **GetLogicalDev (**
pszLogDev As String,
plLogID As Long
) As IUnknown

Parameters:

pszLogDev:

A string containing the logical name of the desired device.

PlLogID:

A variable in which to store the logical ID of the desired device.

Return Values:

Returns an object reference to the device if successful, NULL if an error occurred. Also, the logical ID of the device is returned in the **plLogID** variable.

Remarks:

The Configurator creates and controls the lifetime of all device objects. To gain access to a device you must know the logical name of that device. The Configurator will provide the correct logical ID to use when communicating with the device. You *must* use the logical ID provided by the Configurator or you may inadvertently access other sections of driver code, which may cause unstable behavior.

Use the **Set** statement to assign the object reference to an appropriate variable. This may be any Interface exposed by the desired device. Note: all devices support the **INHRDev** interface.

Release the object by setting the reference variable to **Nothing** when it is no longer needed.

Example:

Obtain a reference to the device named “DMM Measurement 1” in the program, get its sub-type and then properly release the object:

```
Dim obDev As INHRDev
Dim lLogId As Long
Dim sSubType As String

Set obDev = g_obCfgMgr.GetLogicalDev(“DMM Measurement 1”, lLogId)
sSubType = obDev.GetLogicalDevSubType(lLogId)
MsgBox “The sub-type is “ & sSubType, vbInformation
Set obDev = Nothing
```

See Also:

GetLogicalDevSubType

16. MEMBER OF NHRINTERFACESLIB.NHRDATA

16.1 BlobMgr

A pointer to the blob manager.
Member of NHRINTERFACESLib.NHRData

Property: BlobMgr As Object

Parameters:

None

Return Values:

Returns an object reference to the Data Manager's Blob Manager.

Remarks:

This function will retrieve the desired blob from the Blob Manager into the variant parameter.

Use the **Set** statement to assign the object reference to an NHRBlob type variable. This is the only interface exposed by the Blob Manager object.

Release the object by setting the reference variable to **Nothing** when it is no longer needed.

Example:

Obtain a reference to the Data Manager's Blob Manager, use it to determine a blob schema, and then properly release the object:

```
Set obBlobMgr = g_obDataMgr.BlobMgr

If obBlobMgr.Schema(hBlob)) > 1 Then
    MsgBox "Incompatible property data!", vbCritical
End If

Set obBlobMgr = Nothing
```

See Also:

BlobStruct, Schema, BlobLen, RetrieveBlob

17. MEMBER OF NHRINTERFACESLIB.NHRFAULT

17.1 CheckFault

Returns TRUE if a fault has been stored.
Member of NHRINTERFACESLib.NHRFault

Function: CheckFault As Long

Parameters:

None

Return Values:

Returns **True** if a fault has been stored, otherwise **False**.

Remarks:

Use this function any time you want to see if anything has logged a fault during the execution of your code. The more frequently you check, the easier it will be to debug the cause of the failure. At a minimum, check for faults at the end of your test routine algorithm. Faults are reported by passing the information to the dataloggers using the Label function of the Datalog object.

Example:

See if any faults have been logged. If so, log the faults to the dataloggers, clear the fault and increment a failure counter:

```
Dim lFault As Long
Dim lSecondary As Long
Dim sDescription As String
Dim sMoreInfo As String
Dim dateCode As Date

' 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
```

See Also:

GetFault, ClearFault, Label

17.2 ClearFault

Clears the fault. Used after the fault has been resolved.
Member of NHRINTERFACESLib.NHRFault

Sub ClearFault

Parameters:

None

Return Values:

None

Remarks:

Use this function to clear the fault and any stored information after the fault has been reported. Faults are reported by passing the information to the dataloggers using the Label function of the Datalog object. After clearing a fault, subsequent calls to CheckFault will return False until another fault is logged.

Example:

See if any faults have been logged. If so, log the faults to the dataloggers, clear the fault and increment a failure counter:

```
Dim lFault As Long
Dim lSecondary As Long
Dim sDescription As String
Dim sMoreInfo As String
Dim dateCode As Date

' 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
```

See Also:

CheckFault, GetFault, Label

17.3 GetFault

Returns the fault descriptions, time, and fault code of the stored fault. Does not clear the stored fault.
Member of NHRINTERFACESLib.NHRFault

Function: **GetFault (**
plSecondaryFault As Long,
pbstrFaultDescription As String,
pbstrAdditionalInfo As String,
pdDate As Date
) As Long

Parameters:

plSecondaryFault:

A long variable that will be filled in by the GetFault call with the secondary fault code logged during the initial fault log.

pbstrFaultDescription:

A string variable that will be filled in by the GetFault call with a description of the fault.

pbstrAdditionalInfo:

A string variable that will be filled in by the GetFault call with additional information (such as the context of the fault) gathered after the fault was logged.

pdDate:

A date variable that will be filled in by the GetFault call with the day and time the fault was logged.

Return Values:

Returns the value of the logged fault.

Remarks:

This function returns descriptive information about the logged fault without clearing the stored information.

Example:

See if any faults have been logged. If so, log the faults to the dataloggers, clear the fault and increment a failure counter:

```
Dim lFault As Long
Dim lSecondary As Long
Dim sDescription As String
Dim sMoreInfo As String
Dim dateCode As Date

' 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
```

See Also:

CheckFault, ClearFault

18. MEMBER OF NHRINTERFACESLIB.NHRHELPER

18.1 EvaluateValResult

Compare the value against a minimum and maximum and return the result as a RESULT_TYPE.
Member of NHRINTERFACESLib.NHRHelper

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

Parameters:

dLowerExtreme:

The extreme minimum acceptable value of the measurement. If **pdValue** is less than this value, **pdValue** will be set to this value.

dMinimum:

The minimum value in which **pdValue** may be considered to Pass. **pdValue** must be between **dMinimum** and **dMaximum** (inclusive) to pass.

pdValue:

The value to be evaluated. If the software is run offline, or the value is outside the range defined by **dLowerExtreme** and **dUpperExtreme**, this value will be modified by the call.

dMaximum:

The maximum value in which **pdValue** may be considered to Pass. **PdValue** must be between **dMinimum** and **dMaximum** (inclusive) to pass.

dUpperExtreme:

The extreme maximum acceptable value of the measurement. If **pdValue** is greater than this value, **pdValue** will be set to this value.

psResult:

A string variable that will be loaded with a string representation of the returned RESULT_TYPE. This will be either “Pass” of “Fail”.

bAlwaysFail:

Optional parameter (default = False). This parameter is ignored unless the software is being run offline. If True, the function will modify **pdValue** to contain a randomized value outside the range defined by **dMinimum** and **dMaximum**. When False, the function will modify **pdValue** to contain a randomized value either inside or outside the **dMinimum** and **dMaximum** range according to the system’s Pass Percentage setup

pIDispatchSPC:

Reserved parameter (default = NULL). This parameter must be NULL or left to the default.

Return Values:

The return value is PASS_RESULT if **pdValue** is between **dMinimum** and **dMaximum**, inclusively, otherwise it is FAIL_RESULT. In addition, the **psResult** variable is modified to contain either “Pass” or “Fail” as appropriate.

Remarks:

This function is used to evaluate a value-type measurement. Do not pass a numeric value for **pdValue** as the function will modify this parameter if the system is running offline. See the notes in the Parameters section for individual details.

Example:

Setup, initiate and fetch the value of a measurement defined by a measurement vector, and increment a failure count if the value is outside the acceptable minimum and maximum values:

```
Dim dMeasurement As Double
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lFailCount as Long

obDMM.SetupBlobs lLogID, sMuxChannel, hBlob, dRange
obDMM.Meas lLogID
obDMM.Fetch lLogID, dMeasurement

‘ Evaluate the measurement and increase the Fail Count if necessary
lResult = g_obHelper.EvaluateValResult(DBL_MIN, 4.95, dMeasurement, 5.05, DBL_MAX,
sResult)
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

EvaluateStrResult, Setup, SetupBlobs, Meas, Fetch

18.2 SetInputVector

Set each input according to the supplied vector ID.
Member of NHRINTERFACESLib.NHRHelper

Function: **SetInputVector (**
lMode As INPUT_VECTOR_MODE,
lInputVectorId As Long,
[bReversed As Long]
) As Long

Parameters:

lMode:

One of the following operation mode values:

USE_VECTOR_ID

Applies the device states defined by the input vector to hardware

RESET_VOLTAGE

The Voltage property of each Input device is set to zero; the lInputVectorId parameter is ignored.

SET_TO_OFF_STATE

All the Input devices are set to their individually defined off states; the lInputVectorId parameter is ignored.

lInputVectorId:

The record ID of the desired input vector.

bReversed:

Optional parameter (default = False). Set to **True** to set the devices in reverse order.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

This function applies the device states defined by the input vector to hardware. While the normal operation is to set the Inputs left to right according to their order in the PowerEdit Input tab, setting the **bReversed** parameter to **True** will cause the order to be right to left.

The Data Manager maintains the list of all input vectors defined in the program. To gain access to input vectors you must obtain a reference to the **INHRDataInput** interface of the Data Manager. Use the **NumInVectors** property to determine the number of input vectors available, and **GetInVectorId** to obtain the record ID of each vector.

Example:

Apply the first input vector defined in the program to the hardware:

```
Dim lIndex As Long
Dim lInputVectorId As Long

lIndex = 0
lInputVectorId = obDataMgrIn.GetInVectorId(lIndex)
g_obHelper.SetInputVector USE_VECTOR_ID, lInputVectorId
```

See Also:

INPUT_VECTOR_MODE, NumInVectors, GetInVectorId

18.3 SetOutputVector

Set each output according to the supplied vector ID.
Member of NHRINTERFACESLib.NHRHelper

Function: **SetOutputVector (**
lOutputVectorId As Long
) As Long

Parameters:

lOutputVectorId:

The record ID of the desired output vector.

Return Values:

Returns **True** if successful, otherwise **False**.

Remarks:

This function applies the device states defined by the output vector to hardware.

The Data Manager maintains the list of all output vectors defined in the program. To gain access to output vectors you must obtain a reference to the **INHRDataOutput** interface of the Data Manager. Use the **NumOutVectors** property to determine the number of output vectors available, and **GetOutVectorId** to obtain the record ID of each vector.

Example:

Apply the first output vector defined in the program to the hardware:

```
Dim lIndex As Long
Dim lOutputVectorId As Long

lIndex = 0
lOutputVectorId = obDataMgrOut.GetOutVectorId(lIndex)
g_obHelper.SetOutputVector lOutputVectorId
```

See Also:

NumOutVectors, GetOutVectorId

19. MEMBER OF NHRINTERFACESLIB.INHRTRHELPER

19.1 ActiveFixture

Determine whether the A fixture or B fixture is active. Valid only during run mode.
Member of NHRINTERFACESLib.INHRTRHelper

Property: ActiveFixture As FIXTURE_ID

Parameters:

None

Return Values:

The active fixture ID (A_FIXTURE or B_FIXTURE)

Remarks:

The value returned by this property is only valid when called during run mode.

19.2 AlwaysFail

Force all offline comparisons to fail.
Member of NHRINTERFACESLib.INHRTRHelper

Property: AlwaysFail As Boolean

Parameters:

bAlwaysFail:

If **True**, the data evaluation functions will generate a randomized value outside the acceptable range guaranteeing a failure.

Return Values:

This call returns the current state of the property.

Remarks:

The setting of this function is intended for system use only, and is automatically set by *emPower*™ software. The parameter and its functionality are ignored unless the software is being run offline. You may read the setting to determine the state at any time.

See Also:

EvaluateStrResult, EvaluateValResult, LogValue, StrResult, ValResult

19.3 CANProperties

Set the CAN specific communications properties.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub CANProperties (
                    sPortName As String,
                    [lArbitrationID As Long = 0],
                    [lStandardMask As Long = &h7FF],
                    [lStandardComp As Long = &hCFFFFFFF],
                    [lExtendedMask As Long = &h1FFFFFFF],
                    [lExtendedComp As Long = &hCFFFFFFF],
                    [dReadTimeout As Double = 0.1],
                    [dWriteTimeout As Double = 0.1]
                  )
```

Parametersl:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

lArbitrationID:

Optional parameter (default = 0). An 11- or 29-bit ID transmitted as the first field of a CAN frame. The arbitration ID determines the priority of the frame, and is normally used to identify the data transmitted in the frame.

lStandardMask:

Optional parameter (default = &h7FF). A bitmask used in conjunction with **lStandardComp** for filtration of incoming standard CAN frames. For each bit set in the mask, the corresponding bit in the standard frame comparator is checked for a match. Bits in the mask that are clear are treated as don't-cares.

lStandardComp:

Optional parameter (default = &hCFFFFFFF). CAN arbitration ID for the standard frame comparator. This comparator filters all incoming standard (11-bit) CAN frames placed into the read queue. The default value (&hCFFFFFFF) is a special value indicating no CAN arbitration ID is selected (all communications is filtered out).

lExtendedMask:

Optional parameter (default = &h1FFFFFFF). A bitmask used in conjunction with *lExtendedComp* for filtration of incoming extended CAN frames. For each bit set in the mask, the corresponding bit in the extended frame comparator is checked for a match. Bits in the mask that are clear are treated as don't-cares.

lExtendedComp:

Optional parameter (default = &hCFFFFFFF). CAN arbitration ID for the extended frame comparator. This comparator filters all incoming extended (29-bit) CAN frames placed into the read queue. The default value (&hCFFFFFFF) is a special value indicating no CAN arbitration ID is selected (all communications is filtered out).

dReadTimeout:

Optional parameter (default = 0.1 seconds). The amount of time to wait for the UUT to respond with data when a read transmission is executed.

dWriteTimeout:

Optional parameter (default = 0.1 seconds). The amount of time to wait for the UUT to respond with status after a write transmission is executed.

Return Values:

None

Remarks:

This driver is based on the National Instruments NI-CAN™ package. Use this function to setup the CAN communications.

Example:

Set the standard and extended comparators and masks such that all CAN data frames are received by network, and gather the response from a CAN based UUT:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT CAN Communications"
CANSetup sPort, 0, 250000
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, 8
CANProperties sPort, 0, 0, 0, 0, 0
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

CANSetup, COMMInitialize, COMMProperties, SendReceive

19.4 CANSetup

Configure the specified Controller Area Network port.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **CANSetup (**
sPortName As String,
[iPortNumber As Integer = 0],
[lBaudRate As Long = 125000],
[lReadQueueLength As Long = 100],
[lWriteQueueLength As Long = 0],
[bLogWarnings As Boolean = False]
) As Boolean

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

iBoardNumber:

Optional parameter (default = 0). The CAN communications port this object will be attached to. The port name is "CAN" plus the value of this parameter (default = CAN0).

lBaudRate:

Optional parameter (default = 125000). Baud (bits per second) rate at which the communications device operates.

lReadQueueLength:

Optional parameter (default = 100). The number of Data Frames that can be stored prior to reading them out.

lWriteQueueLength:

Optional parameter (default = 0). The number of Data Frames stored prior to writing them to the UUT.

bLogWarnings:

Optional parameter (default = False). Set to **True** to log any warnings encountered as errors. This can be very useful during debugging.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

Use this function to configure the specified CAN port.

Example:

Send information and gather the response from a CAN-based UUT running on port 0 at 250K baud. Accept the default states of the remaining setup parameters:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT CAN Communications"
CANSetup sPort, 0, 250000
COMMInitialize sPort
COMMProperties sPort, False, True, "\xAB\xCD\xEF", True, 8
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

COMMInitialize, COMMProperties, SendReceive

19.5 CheckFault

Checks to see if any faults have been reported. Returns a fault code and a formatted string, if necessary. Member of NHRINTERFACESLib.INHRTRHelper

Function: **CheckFault (**
sFault As String
) As Long

Parameters:

sFault:

A string variable that will be loaded with a fault message if a fault exists. The string will remain unmodified if no faults have been reported.

Return Values:

Returns the stored fault code if a fault has been reported, otherwise the return value is zero.

Remarks:

Use this function any time you want to see if anything has reported a fault during the execution of your code. The more frequently you check, the easier it will be to debug the cause of the failure. At a minimum, check for faults at the end of your test routine algorithm. Faults are logged by passing the information to the dataloggers using the **Label** function.

Example:

See if any faults have been reported. If so, log the faults to the dataloggers and increment a failure counter:

```
Dim lFault As Long
Dim sFault As String

' Check for faults; log any faults found
lFault = obTRHelper.CheckFault(sFault)
If lFault <> 0 Then
    obTRHelper.Label "Error: " & sFault, True
    lFailCount = lFailCount + 1
End If
```

See Also:

Label

19.6 COMMBitMask

Bit mask the value of the returned UUT communications string.
Member of NHRINTERFACESLib.INHRTRHelper

Sub COMMBitMask (
sPortName As String,
bEnable As Boolean,
[lBitMask As Long = 0]
)

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

bEnable:

True to mask the UUT response according to the delimiter value, False to leave the response string intact.

lBitMask:

Optional parameter (default = 0). The mask value to be applied to the response string.

Return Values:

None

Remarks:

Use this function to define the masking action that will be performed on the response string during a **SendReceive** operation.

The communications object translates the UUT response into a 32-bit value and then performs a logical AND function using the value of */BitMask*. The response string must be a single hexadecimal value for this function (e.g., "\xA5"). Use **Tokenize** and/or **COMMTrim** to manipulate the response string as necessary.

Example:

Read a single character from the CAN bus. Assume it's a status byte, and we're interested in the least significant bit. If the bit is set the UUT is on, if the bit is cleared the UUT is off:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE
Dim lStatusByte As Long

sPort = "UUT CAN Communications"
CANSetup sPort, 0, 250000
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, 1
CANProperties sPort, 0, 0, 0, 0, 0
COMMBitMask sPort, True, &h01
SendReceive sPort, sResponse, sResult, sDescription, lResultType
lStatusByte = CLng(sResponse)
If lStatusByte = 0 Then
    Label "UUT is OFF"
Else
    Label "UUT is ON"
End If
```

See Also:

CANSetup, COMMInitialize, COMMProperties, CANProperties, SendReceive

19.7 COMMEvaluation

Perform tests on the returned UUT communications string.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub COMMEvaluation (
        sPortName As String,
        [lTestMethod As TEST_METHOD = NO_TEST],
        [dReference As Double = 0.0],
        [sMinimumValue As String = ""],
        [sMaximumValue As String = ""],
        [lCompareMethod As STRING_TEST = STRTEST_MATCH],
        [bCaseSensitive As Boolean = False],
        [sMinimumString As String = ""],
        [sMaximumString As String = ""],
    )
```

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

lTestMethod

Optional parameter (default = NO_TEST). Must be one of the following constants:

NO_TEST:	Do not test the response string
TEST_AS_STRING:	Test the response string using string compare methods
TEST_AS_VALUE:	Translate the response string into a value and use numerical compare methods

dReference:

Optional parameter (default = 0.0). A numerical reference value used to calculate the actual minimum/maximum value(s) when sMinimumValue and/or sMaximumValue as specified as a percentage of reference (e.g., "110%"). For example, if sMaximumValue is "110%" and dReference is 5.0, the maximum acceptable value will be 5.5.

sMinimumValue:

Optional parameter (default = "", empty string). The minimum acceptable value when testing the response string as a value (lTestMethod = TEST_AS_VALUE). It can be either a hard-coded value (e.g., "5.5"), or a percentage of the dReference value (e.g., "110%").

sMaximumValue:

Optional parameter (default = "", empty string). The maximum acceptable value when testing the response string as a value (lTestMethod = TEST_AS_VALUE). It can be either a hard-coded value (e.g., "5.5"), or a percentage of the dReference value (e.g., "110%").

lCompareMethod:

Optional parameter (default = STRTEST_MATCH). Specifies the type of string comparison to be performed on the response string when lTestMethod is TEST_AS_STRING. Must be one of the following constants:

STRTEST_MATCH:	The response string must match the value of **sMinimumString**.
STRTEST_NOMATCH:	The response string must NOT match the value of **sMinimumString**.
STRTEST_CONTAIN:	The response string must be a substring of **sMinimumString**.
STRTEST_NOCONTAIN:	The response string must NOT be a substring of **sMinimumString**.
STRTEST_INRANGE:	The response string must be within the range defined by **sMinimumString** and **sMaximumString**, inclusive.
STRTEST_NOINRANGE:	The response string must NOT be within the range defined by **sMinimumString** and **sMaximumString**, inclusive.

bCaseSensitive:

Optional parameter (default = False). True to perform a case sensitive comparison of the strings.

sMinimumString:

Optional parameter (default = “”, empty string). The main lexicographic value with which the response string is compared. sMinimumString is used in all string comparisons. It becomes the minimum lexicographic value in which the response string may be considered to Pass when the comparison is a range type. See Remarks.

sMaximumString:

Optional parameter (default = “”, empty string). The maximum lexicographic value in which the response string may be considered to Pass. sMaximumString is used only when the comparison is a range type.

Return Values:

None

Remarks:

Use this function to define the type of testing that will be performed on the response string during a **SendReceive** operation.

When testing the response string as a value, the response is translated into a floating-point value. The response must be in a format such that the driver is able to perform the translation - use **Tokenize** and/or **COMMTrim** to manipulate the response string as necessary. The maximum value supported by the translation is 3.402823466e+38.

When testing the response string as a string, the function will lexicographically evaluate the response. In most cases the comparison is made between the response and **sMinimumString**; **sMaximumString** is ignored. The exceptions to this are when **lCompareMethod** indicates a range comparison. See the notes in the Parameters section for individual details.

Example:

Read the UUT response and verify that it falls within the case-sensitive string range defined by “abC” and “abE”:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String

sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\r\d"
COMMEvaluation sPort, TEST_AS_STRING, 0, "", "", STRTEST_INRANGE, True, "abC",
"abE"
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

The comparison will pass if the UUT response was either “abC”, “abD” or “abE”. Note that “abc”, “abd” and “abe” will fail because the comparison was case-sensitive.

See Also:

SerialSetup, COMMInitialize, COMMProperties, SendReceive

19.8 COMMInitialize

Initialize the specified UUT communications port properties.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **COMMInitialize (**
sPortName As String,
[hBlob As Long = 0]
) As Boolean

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

hBlob:

Optional parameter (default = 0). The handle to the blob defining the actions to be performed by the communications object. If hBlob is zero, the device will be set to perform its default actions.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

Use this function to set the communications object to its default state, then call COMMProperties and/or one of the port-specific property functions (GPIBProperties, CANProperties or I2CProperties) to customize the actions to be performed.

Example:

Gather the response from a GPIB based UUT at address 1:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT GPIB Communications"
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, 100
GPIBProperties sPort, 1
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

COMMProperties, GPIBProperties, SendReceive

19.9 COMMProperties

Set the basic communications properties common to all COMM devices.
Member of NHRINTERFACESLib.INHRTRHelper

Sub COMMProperties (
sPortName As String,
[bFlush As Boolean = False],
[bSendOutput As Boolean = False],
[sOutput As String = ""],
[bGetInput As Boolean = False],
[vTermination As Variant],
[bKeepBinary As Boolean = False]
)

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

bFlush:

Optional parameter (default = False). Set to True to flush the communications port prior to beginning the transfer.

bSendOutput:

Optional parameter (default = False). Set to True to send data (write) to the UUT.

sOutput:

Optional parameter (default = "", empty string). The data to send to the UUT if **bSendOutput** is True.

bGetInput:

Optional parameter (default = False). Set to True to retrieve data (read) from the UUT.

vTermination:

Optional parameter. The termination option to use when retrieving data from the UUT. Either a count specifying a certain number of characters to read, or a character string to match (e.g., "\r\n").

bKeepBinary:

Optional parameter (default = False). Set to True return the received string as a sequence of hexadecimal escape codes instead of printable ASCII (see Remarks).

Return Values:

None

Remarks:

Use this function to define the various actions that will be performed on the communications object during a **SendReceive** operation.

This function accepts various character sequences (case insensitive) as single data values for both **sOutput** and **vTermination**:

\a = &h07 (bell)
\b = &h08 (backspace)
\t = &h09 (tab)
\n = &h0a (linefeed)
\v = &h0b (vertical tab)
\f = &h0c (formfeed)
\r = &h0d (enter, carriage return)
\e = &h1b (escape)
\\ = backslash
\x + the next 2 numeric values = hex code (e.g., \x00 = null)

Each of these codes counts as one character and, like all string data, will be translated into its binary equivalent by the driver before transmission to the UUT. When reading from the UUT the data will be translated into printable ASCII or character escape sequences as necessary unless the **bKeepBinary** option is True. In this case, the returned string will consist entirely of hex escape codes (note that the length of both examples is 16 characters):

bKeepBinary = False: "Hello, World!\r\n\x00"
bKeepBinary = True: "\x48\x65\x6C\x6C\x6F\x2C\x20\x57\x6F\x72\x6C\x64\x21\x0D\x0A\x00"

Example:

Send a greeting to a serial-based UUT running on COM2 at 4800 baud:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, True, "Hello, UUT!\r\n"
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

SerialSetup, COMMInitialize, SendReceive

19.10 COMMTrim

Trim the returned UUT communications string.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub COMMTrim (
                    sPortName As String,
                    [bTrimSpaces As Boolean = False],
                    [lTrimLeft As Long = 0],
                    [lTrimRight As Long = 0]
                )
```

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

bTrimSpaces:

Optional parameter (default = False). True to remove leading and trailing whitespace (newline, space and tab characters) from the UUT response.

lTrimLeft:

Optional parameter (default = 0). The number of leading characters to remove from the UUT response.

lTrimRight:

Optional parameter (default = 0). The number of trailing characters to remove from the UUT response.

Return Values:

None

Remarks:

Use this function to define the trimming actions that will be performed on the response string during a **SendReceive** operation.

Example:

Trim whitespace and the last 5 characters from the UUT response:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\r\d"
COMMTrim sPort, True, 0, 5
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

SerialSetup, COMMInitialize, COMMProperties, SendReceive

19.11 DINMeas

Perform a Digital Input measurement cycle.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **DINMeas (**
sDINName As String,
sThreshold As String
) As DIGITAL_STATE

Parameters:

sDINName:

The name of the Digital Input to be measured.

sThreshold:

The Digital Input reference voltage.

Return Values:

Returns the DIGITAL_STATE measured by the instrument (see Remarks). If an error occurs the return value is DIGITAL_NO_CHANGE.

Remarks:

This function will set up the instrument to take a DIN measurement. Signals above the specified threshold are reported as DIGITAL_HIGH; signals below the threshold are DIGITAL_LOW.

Example:

Verify that the signal connected to “DIN 001” is above 1.4 volts:

```
Dim dState As DIGITAL_STATE
Dim lFailCount As Long

dState = obTRHelper.DINMeas("DIN 001", "1.4")
If dState <> DIGITAL_HIGH
    lFailCount = lFailCount + 1
End If
```

19.12 DMMMeas

Perform a measurement cycle using a DMM.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **DMMMeas (**
sMeasVectorName As String,
sChannelName As String,
[sRange As String = "0.0"],
[sDelay As String = "0.0"],
[sVectorName As String = ""]
) As Double

Parameters:

sMeasVectorName:

The name of the Measurement Vector defining the desired measurement.

sChannelName:

The name of the channel to be measured. This may be either an actual multiplexer channel (e.g., "Chn 001") or the name of an Input or Output Container (e.g., "UUT Out 1").

sRange:

Optional parameter (default = "0.0"). This value is used to set the measurement range of the instrument. It can be either a hard-coded value (e.g., "5.5"), or a percentage of a reference defined by **sChannelName** (e.g., "110%"). Good practice is to select a maximum value that you expect will never, under normal circumstances, be exceeded by the measurement. This will place the instrument is the range most appropriate to the measurement. A value of zero is interpreted as the maximum range capability of the measurement instrument.

sDelay:

Optional parameter (default = 0 seconds). The amount of time (in seconds) to delay before the measurement. This value can be either a hard-coded value (e.g., "100 mS"), or a percentage of the time reference defined by **sChannelName** (e.g., "110%").

sVectorName:

Optional parameter (default = "", empty string). An Input, Output, Analog or Digital Vector which will be set after the instrument is armed but before the measurement is taken. A system trigger will be sent to coincide with this operation.

Return Values:

Returns the value measured by the instrument.

Remarks:

This function will set up the instrument to take the measurement defined in the Measurement Vector. If defined, the Vector named **sVectorName** is applied after the instrument is armed. The measurement is taken after the programmed delay.

sChannelName must be an Input or Output Container in order to program **sRange** or **sDelay** as a percentage of reference. The call will automatically choose the appropriate reference value for **sRange** based on the function defined by **sMeasVectorName**.

Example:

Perform the measurement defined by the Measurement Vector named "DMM DCV" on the multiplexer channel specified by "UUT Out 1". We're not expecting a value above 5.2 volts, so place the instrument in the 5.5V range:

```
Dim dMeasurement As Double
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lFailCount As Long

dMeasurement = obTRHelper.DMMMeas("DMM DCV", "UUT Out 1", "5.5")
lResult = obTRHelper.EvaluateValResult("4.8", dMeasurement, "5. "2, sResult)
obTRHelper.ValResult dMeasurement, lResult, "Volts", "4.8", "5.2", "V", sResult
```

```
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

EvaluateValResult, ValResult

19.13 DMMMeasTest

A complete test routine in one line, this function will take a DMM measurement, compare the result and log the data.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **DMMMeasTest (**
sMeasVectorName As String,
sChannelName As String,
sMinimum As String,
sMaximum As String,
[sLabel As String = ""],
[sRange As String = "0.0"],
[sDelay As String = "0.0"],
[sVectorName As String = ""],
[sLowerExtreme As String = "-1.79769313486231E+308"],
[sUpperExtreme As String = "1.79769313486231E+308"],
[lIndent As Long = 0],
[lParentGroup As Long = 0],
[lSortOrder As Long = 0],
[lhQMon As Long = 0],
[dNomMin As Double = 0.0],
[dNomMax As Double = 0.0]
) As Long

Parameters:

sMeasVectorName:

The name of the Measurement Vector defining the desired measurement.

sChannelName:

The name of the channel to be measured. This may be either an actual multiplexer channel (e.g., "Chn 001") or the name of an Input or Output Container (e.g., "UUT Out 1").

sMinimum:

The minimum acceptable value for the measurement being logged. This value can be either a hard-coded value (e.g., "5.5"), or a percentage of a reference defined by **sChannelName** (e.g., "110%"). This value is placed in the "Minimum" field of the datalog device.

sMaximum:

The maximum acceptable value for the measurement being logged. This value can be either a hard-coded value (e.g., "5.5"), or a percentage of a reference defined by **sChannelName** (e.g., "110%"). This value is placed in the "Maximum" field of the datalog device.

sLabel:

Optional parameter (default = "", empty string). A string defining the value being logged. In the default case (empty string), a label will be generated automatically, consisting of the name of the measurement channel and the measurement vector (e.g., "UUT Out 1 DMM DCV"). This value is placed in the "Label" field of the datalog device.

sRange:

Optional parameter (default = "0.0"). This value is used to set the measurement range of the instrument. It can be either a hard-coded value (e.g., "5.5"), or a percentage of a reference defined by **sChannelName** (e.g., "110%"). Good practice is to select a maximum value that you expect will never, under normal circumstances, be exceeded by the measurement. This will place the instrument is the range most appropriate to the measurement. A value of zero is interpreted as the maximum range capability of the measurement instrument.

sDelay:

Optional parameter (default = "0.0 S"). The amount of time (in seconds) to delay before the measurement. This value can be either a hard-coded value (e.g., "100 mS"), or a percentage of the time reference defined by sChannelName (e.g., "110%").

sVectorName:

Optional parameter (default = “”, empty string). An Input, Output, Analog or Digital Vector which will be set after the instrument is armed but before the measurement is taken. A system trigger will be sent to coincide with this operation.

sLowerExtreme:

Optional parameter (default = “-1.79769313486231E+308”). The extreme minimum acceptable value of the measurement. It can be either a hard-coded value (e.g., “5.5”), or a percentage of a reference defined by sChannelName (e.g., “110%”). If the measured value is less than this value, the measured value will be set to this value.

SUpperExtreme:

Optional parameter (default = “1.79769313486231E+308”). The extreme maximum acceptable value of the measurement. It can be either a hard-coded value (e.g., “5.5”), or a percentage of a reference defined by sChannelName (e.g., “110%”). If the measured value is greater than this value, the measured value will be set to this value.

lIndent:

Optional parameter (default = 0). The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

lParentGroup:

Optional parameter (default = 0). Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Optional parameter (default = 0). Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

hQMon:

Optional parameter (default = 0). Handle to the blob defining the Quality Monitor properties used during this test.

dNomMin:

Optional parameter (default = 0.0). A normalized minimum value used when the value expressed by sMinimum varies due to relative comparison tests. This parameter is only used if Quality Monitor data is being collected and the value of sMinimum is different from its last recorded usage for this Quality Monitor data point.

dNomMax:

Optional parameter (default = 0.0). A normalized maximum value used when the value expressed by sMaximum varies due to relative comparison tests. This parameter is only used if Quality Monitor data is being collected and the value of sMinimum is different from its last recorded usage for this Quality Monitor data point.

Return Values:

Returns the number of failures detected during the execution.

Remarks:

Use this function to perform a single DMM measurement test, including comparison and datalogging functions. It can optionally collect Quality Monitor data on the result.

sChannelName must be an Input or Output Container in order to program **sMinimum**, **sMaximum**, **sRange, sDelay**, **sLowerExtreme** or **sUpperExtreme** as a percentage of reference. The call will automatically choose the appropriate reference value for all references (except **sDelay**) based on the function defined by **sMeasVectorName**. The reference value for **sDelay** is always the time reference.

Example:

Test and log the results of the channel named “UUT Out 1” using the Measurement Vector named “DMM DCV”:

```
lFailCount = lFailCount + obTRHelper.DMMMeasTest("DMM DCV", "UUT Out 1", "4.8",
"5.2", "Volts")
```

This single line is the equivalent of:

```
dMeasurement = obTRHelper.DMMMeas("DMM DCV", "UUT Out 1")
lResult = obTRHelper.EvaluateValResult("4.8", dMeasurement, "5.2", sResult)
sUnits = GetMeasurementUnits("DMM DCV")
obTRHelper.ValResult dMeasurement, lResult, "Volts", "4.8", "5.2", sUnits, sResult
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

DMMMeas, EvaluateValResult, GetMeasurementUnits, ValResult

19.14 EvaluateStrResult

Compare the string against a minimum and maximum and return the result as a RESULT_TYPE.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **EvaluateStrResult (**
sMinimum As String,
sString As String,
sMaximum As String,
sResult As String,
sTestDescription As String,
[lStringTestType As STRING_TEST = STRTEST_MATCH],
[bCaseSensitive As Boolean = False]
) As RESULT_TYPE

Parameters:

sMinimum:

The main lexicographic value with which **sString** is compared. **sMinimum** is used in all string comparisons. It becomes the minimum lexicographic value in which **sString** may be considered to Pass when the comparison is a range type. See Remarks.

sString:

The string to be evaluated. If the software is run offline this value will be modified by the call.

sMaximum:

The maximum lexicographic value in which **sString** may be considered to Pass. **sMaximum** is used only when the comparison is a range type.

sResult:

A string variable that will be loaded with a string representation of the returned RESULT_TYPE. This will be either “Pass” or “Fail”.

sTestDescription:

A string variable that will be loaded with a string representation of **lStringTestType**. For example, if **lStringTestType** is STRTEST_MATCH, **sTestDescription** will contain “Test for match”.

lStringTestType:

Optional parameter (default = STRTEST_MATCH). The type of string compare to perform. It must be one of the following constants:

STRTEST_MATCH:	The value of **sString** must match the value of **sMinimum**.
STRTEST_NOMATCH:	The value of **sString** must NOT match the value of **sMinimum**.
STRTEST_CONTAIN:	The value of **sString** must be a substring of **sMinimum**.
STRTEST_NOCONTAIN:	The value of sString must NOT be a substring of **sMinimum**.
STRTEST_INRANGE:	The value of **sString** must be within the range defined by **sMinimum** and **sMaximum**, inclusive.
STRTEST_NOINRANGE:	The value of **sString** must NOT be within the range defined by **sMinimum** and **sMaximum**, inclusive.

bCaseSensitive:

Optional parameter (default = False). True to perform a case sensitive comparison of the strings.

Return Values:

The return value is PASS_RESULT if **sString** is lexicographically equal to (or contained by, etc.) **sMinimum** (and **sMaximum** if a range comparison), otherwise it is FAIL_RESULT. In addition, the **sResult** variable is modified to contain either “Pass” or “Fail” as appropriate, and **sTestDescription** is modified to contain a string describing the type of comparison made (according to the value of **lStringTestType**).

Remarks:

This function is used to lexicographically evaluate a string-type measurement. In most cases the comparison is made between **sString** and **sMinimum**; **sMaximum** is ignored. The exceptions to

this are when **lStringTestType** indicates a range comparison. See the notes in the Parameters section for individual details.

Example:

Assume a UUT Communications test returned the string assigned to **sMeasurement**. Increment a failure count if the value does not contain the expected substring “3ab”. (Note that this example will pass because the case insensitive search for the substring is found in the string “123ABC”):

```
Dim sMeasurement As String
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim sTestDesc As String
Dim lFailCount As Long

sMeasurement = "123ABC"
lResult = obTRHelper.EvaluateStrResult("3ab", sMeasurement, "", sResult,
sTestDesc, STRTEST_CONTAIN)
obTRHelper.StrResult sMeasurement, lResult, "UUT Communications " & sTestDesc,
"3ab", "", sResult
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

EvaluateValResult, StrResult

19.15 EvaluateValResult

Compare the value against a minimum and maximum and return the result as a RESULT_TYPE.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **EvaluateValResult (**
sMinimum As String,
dValue As Double,
sMaximum As String,
sResult As String,
[sLowerExtreme As String = "-1.79769313486231E+308"],
[sUpperExtreme As String = "1.79769313486231E+308"]
) As RESULT_TYPE

Parameters:

sMinimum:

The minimum value in which *dValue* may be considered to Pass. **sMinimum** will be translated to a double value prior to comparison with **dValue**. **dValue** must be between **sMinimum** and **sMaximum** (inclusive) to pass.

dValue:

The value to be evaluated. If the software is run offline, or the value is outside the range defined by **sLowerExtreme** and **sUpperExtreme**, this value will be modified by the call.

sMaximum:

The maximum value in which **dValue** may be considered to Pass. **sMinimum** will be translated to a double value prior to comparison with **dValue**. **dValue** must be between **sMinimum** and **sMaximum** (inclusive) to pass.

sResult:

A string variable that will be loaded with a string representation of the returned RESULT_TYPE. This will be either "Pass" or "Fail".

sLowerExtreme:

Optional parameter (default = "-1.79769313486231E+308"). The extreme minimum acceptable value of the measurement. If **dValue** is less than this value, **dValue** will be set to this value.

sUpperExtreme:

Optional parameter (default = "1.79769313486231E+308"). The extreme maximum acceptable value of the measurement. If **dValue** is greater than this value, **dValue** will be set to this value.

Return Values:

The return value is PASS_RESULT if **dValue** is between **sMinimum** and **sMaximum**, inclusively, otherwise it is FAIL_RESULT. In addition, the **sResult** variable is modified to contain either "Pass" or "Fail" as appropriate.

Remarks:

This function is used to evaluate a value-type measurement. Do not pass a numeric value for **dValue** as the function will modify this parameter if the system is running offline. See the notes in the Parameters section for individual details.

Example:

Take a measurement defined by a DMM measurement vector, and increment a failure count if the value is outside the acceptable minimum and maximum values:

```
Dim dMeasurement As Double
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lFailCount As Long

dMeasurement = obTRHelper.DMMMeas("DMM DCV", "Chn 001", "5.5")
lResult = obTRHelper.EvaluateValResult("4.8", dMeasurement, "5.2", sResult)
obTRHelper.ValResult dMeasurement, lResult, "Volts", "4.8", "5.2", "V", sResult
If lResult = FAIL_RESULT
```

```
        lFailCount = lFailCount + 1
    End If
```

See Also:

EvaluateStrResult, DMMMeas, ValResult

19.16 Flush

Immediately flushes the UUT communication buffers.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub Flush (
            sPortName As String
          )
```

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

Return Values:

None

Remarks:

The action performed is dependent on the bus:

Serial: discards all characters from the input and output buffer of the specified port
GPIB: no operation
CAN: discards the contents of the read queue
I^2C: no operation

The flush can also be performed automatically with **SendReceive** by setting **COMMProperties**::**bFlush** to True.

Example:

Flush the serial port defined by “UUT Communications”:

```
Flush "UUT Communications"
```

See Also:

SendReceive, COMMProperties

19.17 GetAnalogName

Gets the name of the Analog device by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetAnalogName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Analog device.

Return Values:

Returns the user-defined description of the Analog device.

Remarks:

Use the **NumAnalogs** property to determine the number of Analog devices available.

Example:

Get the name of each Analog device configured in the program, then obtain an object reference for the Analog and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obAnalog As INHRAnalog
Dim lLogID As Long

For lCount = 0 To obTRHelper.NumAnalogs - 1
    sName = obTRHelper.GetAnalogName(lCount)
    Set obAnalog = g_obCfgMgr.GetLogicalDev(sName, lLogID)
    ' Do something with the object
    Set obAnalog = Nothing
Next
```

See Also:

NumAnalogs

19.18 GetAnalogVectorName

Gets the name of the Analog Vector by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetAnalogVectorName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Analog Vector.

Return Values:

Returns the user-defined description of the Analog Vector.

Remarks:

Use the **NumAnalogVectors** property to determine the number of Analog Vectors defined.

Example:

Get the name of each Analog Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumAnalogVectors - 1
    sName = obTRHelper.GetAnalogVectorName(lCount)
Next
```

See Also:

NumAnalogVectors

19.19 GetDigitalName

Gets the name of the Digital Container by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetDigitalName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Digital Container.

Return Values:

Returns the user-defined description of the Digital Container.

Remarks:

Use the **NumDigitals** property to determine the number of Digital Containers available.

Example:

Get the name of each Digital Container configured in the program, then obtain an object reference for the Digital and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumDigitals - 1
    sName = obTRHelper.GetDigitalName(lCount)
    Set obContainer = obTRHelper.GetDigitalObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

NumDigitals, GetDigitalObj

19.20 GetDigitalObj

Gets a reference to the Digital Container by name.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetDigitalObj (**
sName As String
) As Variant

Parameters:

sName:

The name of the desired Digital Container.

Return Values:

Returns a reference to the Digital Container.

Remarks:

The object reference is returned as a variant. You must assign the variant to an appropriate object variable before use. Valid object variables are: **NHRDigtalContainer**, **INHRContainer**.

Example:

Get the name of each Digital Container configured in the program, then obtain an object reference for the Digital and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumDigitals - 1
    sName = obTRHelper.GetDigitalName(lCount)
    Set obContainer = obTRHelper.GetDigitalObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

NumDigitals, GetDigitalName

19.21 GetDigitalVectorName

Gets the name of the Digital Vector by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetDigitalVectorName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Digital Vector.

Return Values:

Returns the user-defined description of the Digital Vector.

Remarks:

Use the **NumDigitalVectors** property to determine the number of Digital Vectors defined.

Example:

Get the name of each Digital Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumDigitalVectors - 1
    sName = obTRHelper.GetDigitalVectorName(lCount)
Next
```

See Also:

NumDigitalVectors

19.22 GetInputName

Gets the name of the Input Container by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetInputName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Input Container.

Return Values:

Returns the user-defined description of the Input Container.

Remarks:

Use the **NumInputs** property to determine the number of Input Containers available.

Example:

Get the name of each Input Container configured in the program, then obtain an object reference for the Input and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumInputs - 1
    sName = obTRHelper.GetInputName(lCount)
    Set obContainer = obTRHelper.GetInputObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

NumInputs, GetInputObj

19.23 GetInputObj

Gets a reference to the Input Container by name.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetInputObj (**
sName As String
) As Variant

Parameters:

sName:

The name of the desired Input Container.

Return Values:

Returns a reference to the Input Container.

Remarks:

The object reference is returned as a variant. You must assign the variant to an appropriate object variable before use. Valid object variables are: **NHRInputContainer**, **INHRContainer**, **INHRContainerMux**.

Example:

Get the name of each Input Container configured in the program, then obtain an object reference for the Input and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumInputs - 1
    sName = obTRHelper.GetInputName(lCount)
    Set obContainer = obTRHelper.GetInputObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

NumInputs, GetInputName

19.24 GetInVectorName

Gets the name of the Input Vector by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetInVectorName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Input Vector.

Return Values:

Returns the user-defined description of the Input Vector.

Remarks:

Use the **NumInVectors** property to determine the number of Input Vectors defined.

Example:

Get the name of each Input Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumInVectors - 1
    sName = obTRHelper.GetInVectorName(lCount)
Next
```

See Also:

NumInVectors

19.25 GetMeasurementUnits

Get the units descriptor of the named Measurement Vector.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetMeasurementUnits (**
sMeasVectorName As String
) As String

Parameters:

sMeasVectorName:

The name of the Measurement Vector defining the desired measurement.

Return Values:

Returns the units descriptor of the named Measurement Vector (e.g., “V”, “Hz”, etc.).

Remarks:

Use this function to get the units string required by **ValResult**.

Example:

Get the measurement units for the Measurement Vector named “DMM DCV” and log them along with the measured data:

```
Dim dMeasurement As Double
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim sUnits As String
Dim lFailCount As Long

dMeasurement = obTRHelper.DMMMeas("DMM DCV", "UUT Out 1", "5.5")
lResult = obTRHelper.EvaluateValResult("4.8", dMeasurement, "5.2", sResult)
sUnits = obTRHelper.GetMeasurementUnits("DMM DCV")
obTRHelper.ValResult dMeasurement, lResult, "Volts", "4.8", "5.2", sUnits, sResult
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

DMMMeas, EvaluateValResult, ValResult

19.26 GetMeasVectorName

Gets the name of the Measurement Vector by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetMeasVectorName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Measurement Vector.

Return Values:

Returns the user-defined description of the Measurement Vector.

Remarks:

Use the **NumMeasVectors** property to determine the number of Measurement Vectors defined.

Example:

Get the name of each Measurement Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumMeasVectors - 1
    sName = obTRHelper.GetMeasVectorName(lCount)
Next
```

See Also:

NumMeasVectors

19.27 GetOutputName

Gets the name of the Output Container by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetOutputName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Output Container.

Return Values:

Returns the user-defined description of the Output Container.

Remarks:

Use the **NumOutputs** property to determine the number of Output Containers available.

Example:

Get the name of each Output Container configured in the program, then obtain an object reference for the Output and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumOutputs - 1
    sName = obTRHelper.GetOutputName(lCount)
    Set obContainer = obTRHelper.GetOutputObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

NumOutputs, GetOutputObj

19.28 GetOutputObj

Gets a reference to the Output Container by name.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetOutputObj (**
sName As String
) As Variant

Parameters:

sName:

The name of the desired Output Container.

Return Values:

Returns a reference to the Output Container.

Remarks:

The object reference is returned as a variant. You must assign the variant to an appropriate object variable before use. Valid object variables are: **NHROutputContainer**, **INHRContainer**, **INHRContainerMux**.

Example:

Get the name of each Output Container configured in the program, then obtain an object reference for the Output and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumOutputs - 1
    sName = obTRHelper.GetOutputName(lCount)
    Set obContainer = obTRHelper.GetOutputObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

NumOutputs, GetOutputName

19.29 GetOutVectorName

Gets the name of the Output Vector by index (0-based).
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetOutVectorName (**
lIndex As Long
) As String

Parameters:

lIndex:

The zero-based index of the desired Output Vector.

Return Values:

Returns the user-defined description of the Output Vector.

Remarks:

Use the **NumOutVectors** property to determine the number of Output Vectors defined.

Example:

Get the name of each Output Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumOutVectors - 1
    sName = obTRHelper.GetOutVectorName(lCount)
Next
```

See Also:

NumOutVectors

19.30 GetStringToken

Get a string token from the UUT communications response.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GetStringToken (**
sPortName As String
) As String

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

Return Values:

The token indexed by **lTokenNumber**.

Remarks:

Use this function to retrieve any of the tokens found in the UUT response.

Example:

Tokenize the received data using a “space” character as the delimiter, and collect each token:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE
Dim sToken() As String
Dim lToken As Long
Dim lCount As Long

sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\r\d"
Tokenize sPort, True, " ", 3
SendReceive sPort, sResponse, sResult, sDescription, lResultType

lCount = NumTokens(sPort)
ReDim sToken(1 To lCount)
For lToken = 1 To lCount
    sToken(lCount) = GetStringToken(sPort, lToken)
Next lToken
```

See Also:

SerialSetup, COMMInitialize, COMMProperties, Tokenize, SendReceive, NumTokens

19.31 GPIBProperties

Set the GPIB specific communications properties.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub GPIBProperties (
                    sPortName As String,
                    [lPrimaryAddress As Long = 0],
                    [lSecondaryAddress As Long = 0],
                    [lTimeoutCode As GPIB_TIMEOUT = GPIB_NO_TIMEOUT]
                  )
```

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

lPrimaryAddress:

Optional parameter (default = 0). Sets the primary GPIB address of the device.

lSecondaryAddress:

Optional parameter (default = 0). Sets the secondary GPIB address of the device. If zero (default), secondary addressing is disabled.

lTimeoutCode:

Optional parameter (default = GPIB_NO_TIMEOUT). Sets the timeout period of the device. The timeout period is used to select the maximum duration allowed for a synchronous I/O operation. If the operation does not complete before the timeout period elapses, then the operation is aborted. Must be one of the following constants:

GPIB_NO_TIMEOUT:	No timeout (default).
GPIB_10US:	10 microseconds
GPIB_30US:	30 microseconds
GPIB_100US:	100 microseconds
GPIB_300US:	300 microseconds
GPIB_1MS:	1 millisecond
GPIB_3MS:	3 milliseconds
GPIB_10MS:	10 milliseconds
GPIB_30MS:	30 milliseconds
GPIB_100MS:	100 milliseconds
GPIB_300MS:	300 milliseconds
GPIB_1S:	1 second
GPIB_3S:	3 seconds
GPIB_10S:	10 seconds
GPIB_30S:	30 seconds
GPIB_100S:	100 seconds

Return Values:

None

Remarks:

This driver is based on the National Instruments NI-GPIB™ package. Use this function to setup the GPIB communications.

Example:

Specify that the selected GPIB board is GPIB0, set the primary address to 22, disable secondary addressing, set the timeout to 10 milliseconds, and gather the response from a GPIB based UUT:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE
```

```
sPort = "UUT GPIB Communications"
GPIBSetup sPort
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, 100
GPIBProperties sPort, 22, 0, GPIB_10MS
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

GPIBSetup, COMMInitialize, COMMProperties, SendReceive

19.32 GPIBSetup

Configure the specified General Purpose Interface Bus port.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **GPIBSetup (**
sPortName As String,
[iBoardNumber As Integer = 0]
) As Boolean

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

iBoardNumber:

Optional parameter (default = 0). The GPIB communications port this object will be attached to. The port name is "GPIB" plus the value of this parameter (default = GPIB0).

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

Use this function to configure the specified GPIB port.

Example:

Specify that the selected GPIB board is GPIB0 and gather the response from a GPIB based UUT at address 1:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT GPIB Communications"
GPIBSetup sPort
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\x0D"
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

COMMInitialize, COMMProperties, SendReceive

19.33 I2CProperties

Set the I^2C specific communications properties.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub I2CProperties (
                    sPortName As String,
                    [lClockRate As Long = CLK_90KHZ],
                    [bMasterMode As Boolean = True],
                    [iSlaveAddress As Integer = 2],
                    [bReadOffset As Boolean = False],
                    [iReadOffsetAddr As Integer = 0],
                    [bWriteOffset As Boolean = False],
                    [iWriteOffsetAddr As Integer = 0],
                    [dReadTimeout As Double = 1.0],
                    [dWriteTimeout As Double = 1.0]
                  )
```

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

lClockRate:

Optional parameter (default = 0). The clock rate of the I^2C board. Must be one of the following constants:

CLK_90KHZ:	90KHz clock rate
CLK_45KHZ:	45KHz clock rate
CLK_11KHZ:	11KHz clock rate
CLK_1_5KHZ:	1.5KHz clock rate

bMasterMode:

Optional parameter (default = True). True to perform a Master Write transmission when sending data, or to perform a Master Read transmission when getting a UUT response.

iSlaveAddress:

Optional parameter (default = 2). A 7-bit address uniquely identifying the UUT (the slave) that you will be reading or writing to. The driver converts this into an eight bit address by adding the read / write bit as required.

bReadOffset:

Optional parameter (default = False). True to enable the use of the offset address specified in **iReadOffsetAddr**.

iReadOffsetAddr:

Optional parameter (default = 0). The offset is an 8-bit address specifying where in the device you want to read.

bWriteOffset:

Optional parameter (default = False). True to enable the use of the offset address specified in **iWriteOffsetAddr**.

iWriteOffsetAddr:

Optional parameter (default = 0). The offset is an 8-bit address specifying where in the device you want to write.

dReadTimeout;

Optional parameter (default = 1.0 second). The amount of time to wait for the UUT to respond with data when a read transmission is executed.

dWriteTimeout:

Optional parameter (default = 1.0 second). The amount of time to wait for the UUT to respond with status after a write transmission is executed.

Return Values

None

Remarks:

This driver is based on the Calibre UK Ltd. I^2C package. Use this function to setup the I^2C communications. The driver supports 4 modes: Master Read, Slave Receiver, Master Write and Slave Transmitter, controlled as follows:

bMasterMode	Reading data	Writing data
True	Master Read	Master Write
False	Slave Receiver	Slave Transmitter

Example:

Place the controller in Master Write mode at 90KHz and send some data to the UUT at slave address 50:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT I2C Communications"
COMMInitialize sPort
COMMProperties sPort, False, True, "\xAB\xCD\xEF"
I2CProperties sPort, CLK_90KHZ, True, 50
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

COMMInitialize, COMMProperties, SendReceive

19.34 Initialize

Performs vital component initialization. Must be called once after instantiation.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub Initialize (
                FaultMgr As Unknown,
                DatalogSvr As Unknown,
                TestInfoSvr As Unknown
              )
```

Parameters:

FaultMgr:

A pointer to the IUnknown interface of the Fault Manger.

DatalogSvr:

A pointer to the IUnknown interface of the Datalog server.

TestInfoSvr:

A pointer to the IUnknown interface of the Test Information server.

Return Values:

None

Remarks:

This function is intended for system use only, and is automatically set by *em**Power***™ software.

19.35 Label

Log a label. Also used to log faults.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **Label (**
sLabel As String,
[bShowAsFault as Boolean = False],
[lIndent As Long = 0],
[lParentGroup As Long = 0],
[lSortOrder As Long = 0]
) As Long

Parameters:

sLabel:

A string defining the value being logged. This value is placed in the "Label" field of the datalog device.

bShowAsFault:

Optional parameter (default = False). True to cause the dataloggers to format the *sLabel* string as appropriate to a fault message.

lIndent:

Optional parameter (default = 0). The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

lParentGroup:

Optional parameter (default = 0). Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Optional parameter (default = 0). Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

Return Values:

Returns a new Parent Group index that can be used as the **lParentGroup** parameter in subsequent datalog calls.

Remarks:

This command logs the supplied label strings to all the configured dataloggers. This command is used to log informational text to the dataloggers. This informational text may be interpreted as a fault message if the **bShowAsFault** parameter is True. Note: logging a fault does not cause the routine or the UUT to fail; it simply informs the dataloggers to format the string as appropriate to a fault message.

Within the same test routine, multiple values logged with the same **lParentGroup** number are grouped together and sub-sorted according to **lSortOrder.** In the case of duplicate **lSortOrder** values, the values are logged in the order they were received. You may pass zero values for both sorting parameters to log all measurement in the order they were received. **lParentGroup** and **lSortOrder** parameters are not maintained across separate test routine executions. Refer to the example under "ValResult".

Example:

Log a Fault to the dataloggers:

```
Dim sFault As String
Dim lFailCount As Long

If obTRHelper.CheckFault(sFault) Then
    obTRHelper.Label "Error: " & sFault, True
    lFailCount = lFailCount + 1
End If
```

Sample Screen driver output:

Label	Minimum	Actual	Maximum	Result
Error: NHR Invalid Parameter.; AC Source 1: SetVoltage: Illegal source voltage: 500				

See Also:

CheckFault

19.36 LogParametrics

Log the set values of the Input and Output Containers.
Member of NHRINTERFACESLib.INHRTRHelper

Property LogParametrics As Boolean

Parameters:

bLogParametrics:

If **True**, the set values of Input and Output Containers are logged to the datalog drivers whenever a Input or Output Vector is set. If **False**, no parametric data is logged.

Return Values:

This call returns the current state of the property.

Remarks:

The setting of this function is intended for system use only, and is automatically set by *em**Power***™ software. You may read the setting to determine the state at any time.

19.37 LogValue

Evaluate a measurement and send the results to the dataloggers.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **LogValue (**
sMeasVectorName As String,
sMinimum As String,
dActual As Double,
sMaximum As String,
[sLabel As String = ""],
[sLowerExtreme As Double = DBL_MIN],
[sUpperExtreme As Double = DBL_MAX],
[lIndent As Long = 0],
[lParentGroup As Long = 0],
[lSortOrder As Long = 0],
[lhQMon As Long = 0],
[dNomMin As Double = 0.0],
[dNomMax As Double = 0.0]
) As Long

Parameters:

sMeasVectorName:

The name of the Measurement Vector defining the desired measurement.

sMinimum:

The minimum acceptable value for the measurement being logged. This value is placed in the "Minimum" field of the datalog device.

dActual:

The numeric value being logged. This value is placed in the "Actual" field of the datalog device.

sMaximum:

The maximum acceptable value for the measurement being logged. This value is placed in the "Maximum" field of the datalog device.

sLabel:

A string defining the value being logged. This value is placed in the "Label" field of the datalog device. May be an empty string if no label is desired.

sLowerExtreme:

Optional parameter (default = "-1.79769313486231E+308"). The extreme minimum acceptable value of the measurement. If *dActual* is less than this value, *dActual* will be set to this value.

sUpperExtreme:

Optional parameter (default = "1.79769313486231E+308"). The extreme maximum acceptable value of the measurement. If **dActual** is greater than this value, **dActual** will be set to this value.

lIndent:

Optional parameter (default = 0). The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

lParentGroup:

Optional parameter (default = 0). Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Optional parameter (default = 0). Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

hQMon:

Optional parameter (default = 0). Handle to the blob defining the Quality Monitor properties used during this test.

dNomMin:

Optional parameter (default = 0.0). A normalized minimum value used when the value expressed by sMinimum varies due to relative comparison tests. This parameter is only used if Quality Monitor data is being collected and the value of **sMinimum** is different from its last recorded usage for this Quality Monitor data point.

dNomMax:

Optional parameter (default = 0.0). A normalized maximum value used when the value expressed by **sMaximum** varies due to relative comparison tests. This parameter is only used if Quality Monitor data is being collected and the value of **sMinimum** is different from its last recorded usage for this Quality Monitor data point.

Return Values:

Returns the number of failures detected during the evaluation.

Remarks:

Use this function to compare a value against a minimum and a maximum and log the results. In addition, the results may be sent to the Quality Monitor for statistical calculations.

Example:

Test and log the results of a measurement:

```
lFailCount = lFailCount + obTRHelper.LogValue("DMM DCV", 4.8, dMeasurement, 5.2,
"Volts")
```

This single line is the equivalent of:

```
lResult = obTRHelper.EvaluateValResult("4.8", dMeasurement, "5.2", sResult)
sUnits = GetMeasurementUnits("DMM DCV")
obTRHelper.ValResult dMeasurement, lResult, "Volts", "4.8", "5.2", sUnits, sResult
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

EvaluateValResult, GetMeasurementUnits, ValResult

19.38 NumAnalogs

Returns the number of Analog devices.
Member of NHRINTERFACESLib.INHRTRHelper

Property NumAnalogs As Long

Parameters:

None

Return Values:

Returns the number of Analog devices configured in the test program.

Remarks:

Use the **NumAnalogs** property to determine the number of Analog devices available.

Example:

Get the name of each Analog device configured in the program, then obtain an object reference for the Analog and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obAnalog As INHRAnalog
Dim lLogID As Long

For lCount = 0 To obTRHelper.NumAnalogs - 1
    sName = obTRHelper.GetAnalogName(lCount)
    Set obAnalog = g_obCfgMgr.GetLogicalDev(sName, lLogID)
    ' Do something with the object
    Set obAnalog = Nothing
Next
```

See Also:

GetAnalogName

19.39 NumAnalogVectors

Returns the number of Analog Vectors defined in the test program.
Member of NHRINTERFACESLib.INHRTRHelper

Property NumAnalogVectors As Long

Parameters:

None

Return Values:

Returns the number of Analog Vectors defined in the test program.

Example:

Get the name of each Analog Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumAnalogVectors - 1
    sName = obTRHelper.GetAnalogVectorName(lCount)
Next
```

See Also:

GetAnalogVectorName

19.40 NumDigitals

Returns the number of Digital Containers.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumDigitals As Long

Parameters:

None

Return Values:

Returns the number of Digital Containers configured in the test program.

Remarks:

Use the **NumDigitals** property to determine the number of Digital Containers available.

Example:

Get the name of each Digital Container configured in the program, then obtain an object reference for the Digital and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumDigitals - 1
    sName = obTRHelper.GetDigitalName(lCount)
    Set obContainer = obTRHelper.GetDigitalObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

GetDigitalName, GetDigitalObj

19.41 NumDigitalVectors

Returns the number of Digital Vectors defined in the test program.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumDigitalVectors As Long

Parameters:

None

Return Values:

Returns the number of Digital Vectors defined in the test program.

Example:

Get the name of each Digital Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumDigitalVectors - 1
    sName = obTRHelper.GetDigitalVectorName(lCount)
Next
```

See Also:

GetDigitalVectorName

19.42 NumInputs

Returns the number of Input Containers.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumInputs As Long

Parameters:

None

Return Values:

Returns the number of Input Containers configured in the test program.

Remarks:

Use the **NumInputs** property to determine the number of Input Containers available.

Example:

Get the name of each Input Container configured in the program, then obtain an object reference for the Input and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumInputs - 1
    sName = obTRHelper.GetInputName(lCount)
    Set obContainer = obTRHelper.GetInputObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

GetInputName, GetInputObj

19.43 NumInVectors

Returns the number of Input Vectors defined in the test program.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumInVectors As Long

Parameters:

None

Return Values:

Returns the number of Input Vectors defined in the test program.

Example:

Get the name of each Input Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumInVectors - 1
    sName = obTRHelper.GetInVectorName(lCount)
Next
```

See Also

GetInVectorName

19.44 NumMeasVectors

Returns the number of Measurement Vectors defined in the test program.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumMeasVectors As Long

Parameters:

None

Return Values:

Returns the number of Measurement Vectors defined in the test program.

Example:

Get the name of each Measurement Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumMeasVectors - 1
    sName = obTRHelper.GetMeasVectorName(lCount)
Next
```

See Also:

GetMeasVectorName

19.45 NumOutputs

Returns the number of Output Containers.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumOutputs As Long

Parameters:

None

Return Values:

Returns the number of Output Containers configured in the test program.

Remarks:

Use the **NumOutputs** property to determine the number of Output Containers available.

Example:

Get the name of each Output Container configured in the program, then obtain an object reference for the Output and do something useful. Be sure to release the object by setting the reference variable to **Nothing** when it is no longer needed:

```
Dim lCount As Long
Dim sName As String
Dim obContainer As INHRContainer

For lCount = 0 To obTRHelper.NumOutputs - 1
    sName = obTRHelper.GetOutputName(lCount)
    Set obContainer = obTRHelper.GetOutputObj(sName)
    ' Do something with the object
    Set obContainer = Nothing
Next
```

See Also:

GetOutputName, GetOutputObj

19.46 NumOutVectors

Returns the number of Output Vectors defined in the test program.
Member of NHRINTERFACESLib.INHRTRHelper

Property: NumOutVectors As Long

Parameters:

None

Return Values:

Returns the number of Output Vectors defined in the test program.

Example:

Get the name of each Output Vector configured in the program:

```
Dim lCount As Long
Dim sName As String

For lCount = 0 To obTRHelper.NumOutVectors - 1
    sName = obTRHelper.GetOutVectorName(lCount)
Next
```

See Also:

GetOutVectorName

19.47 NumTokens

Returns the number of tokens found in the UUT communications string.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **NumTokens (**
sPortName As String
) As Long

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

Return Values:

Returns the number of tokens found in the UUT response.

Remarks:

Use this function along with **GetStringToken** to retrieve any of the tokens found in the UUT response.

Example:

Tokenize the received data using a “space” character as the delimiter, and collect each token:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE
Dim sToken() As String
Dim lToken As Long
Dim lCount As Long

sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\r\d"
Tokenize sPort, True, " ", 3
SendReceive sPort, sResponse, sResult, sDescription, lResultType

lCount = NumTokens(sPort)
ReDim sToken(1 To lCount)
For lToken = 1 To lCount
    sToken(lCount) = GetStringToken(sPort, lToken)
Next lToken
```

See Also:

SerialSetup, COMMInitialize, COMMProperties, Tokenize, SendReceive, GetStringToken

19.48 SendReceive

Send and/or receive UUT communication data.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **SendReceive (**
sPortName As String,
sResponse As String,
sResult As String,
sDescription As String,
lResultType As RESULT_TYPE
) As Boolean

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *emPower*™ program.

sResponse:

A string variable that will be loaded with response string received from the UUT.

sResult:

A string variable that will be loaded with a string representation of the returned RESULT_TYPE. This will be either "Pass", "Fail" or "Untested".

sDescription;

A string variable that will be loaded with a string representation of the type of test performed. For example, if **COMMEvaluation**::**lCompareMethod** is STRTEST_MATCH, **sDescription** will contain "Test for match"; if **COMMEvaluation**::**lTestMethod** is TEST_AS_VALUE, **sDescription** will contain "Test for value".

lResultType:

A RESULT_TYPE variable that will contain the results of any testing performed on the UUT response. Will be one of the following values:

PASS_RESULT
FAIL_RESULT
UNTESTED_RESULT

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

The SendReceive communications procedure is the same regardless of the bus type (Serial, GPIB, CAN or I^2C):

1. Flush input buffer if **COMMProperties**::**bFlush** is True
2. Send string if **COMMProperties**::**bSendOutput** is True
3. Get a response by reading until the termination option is detected if **COMMProperties**::**bGetInput** is True
4. Tokenize the response if **Tokenize**::**bEnable** is True
5. Trim whitespace if **COMMTrim**::**bTrimSpaces** is True
6. Trim extra data from either end according to the settings of **COMMTrim**::**lTrimLeft** and **COMMTrim**::**lTrimRight**
7. Apply the bitmask if **COMMBitMask**::**bEnable** is True
8. Test the response according to the setting of **COMMEvaluation**::**lTestMethod**

Example:

Specify that the selected GPIB board is GPIB0 and gather the response from a GPIB based UUT at address 1:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT GPIB Communications"
GPIBSetup sPort
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\x0D"
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

COMMInitialize, COMMProperties, COMMEvaluation, COMMTrim, Tokenize, GPIBSetup

19.49 SerialSetup

Configure the specified serial port.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **SerialSetup (**
sPortName As String,
[iPortNumber As Integer = 1],
[lBaudRate As Long = 9600],
[lDeviceControl As Long = &h3011],
[iXONLimit As Integer = 10],
[iXOFFLimit As Integer = 10],
[iByteSize As Integer = 8],
[lParity As SERIAL_PARITY = NO_PARITY],
[lStopBits As SERIAL_STOPBITS = ONE_STOPBIT],
[sXON As String = ""],
[sXOFF As String = ""],
[sError As String = ""],
[sEOF As String = ""],
[sEvent As String = ""],
[lInputBufSize As Long = 1024],
[lOutputBufSize As Long = 512],
[dReadIntervalTimeout As Double = 0.1],
[dReadTotalTimeoutMult As Double = 0.1],
[dReadTotalTimeoutConst As Double = 0.1],
[dWriteTotalTimeoutMult As Double = 0.1],
[dWriteTotalTimeoutConst As Double = 0.1]
) As Boolean

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

iPortNumber:

Optional parameter (default = 1). The serial communications port this object will be attached to. The port name is "COM" plus the value of this parameter (default = COM1).

lBaudRate:

Optional parameter (default = 9600). Baud (bits per second) rate at which the communications device operates.

lDeviceControl:

Optional parameter (default = &h3011). A bit field that sets up the basic port configuration.

Bit 0: fBinary:	Reserved. Must be set (1).
Bit 1: fParity:	Specifies if parity checking is enabled. If this member is set, parity checking is performed and errors are reported.
Bit 2: fOutxCtsFlow:	Turns the CTS flow control on and off. To use RTS/CTS flow control, set this member and choose RTS_CONTROL_HANDSHAKE for the **fRtsControl** member.
Bit 3: fOutxDsrFlow:	Turns the DSR flow control on and off. DSR flow control is rarely used. A typical port configuration is to clear this member (0), while enabling the DTR line.
Bits 4-5: fDtrControl:	Specifies the DTR flow control. DTR_CONTROL_ENABLE turns on the DTR line during the connection. DTR_CONTROL_HANDSHAKE turns on DTR handshaking. DTR_CONTROL_DISABLE turns off the DTR line.

DTR_CONTROL_DISABLE: &h00
DTR_CONTROL_ENABLE: &h01
DTR_CONTROL_HANDSHAKE: &h02

Bit 6: fDsrSensitivity: Specifies if the communications driver is sensitive to the state of the DSR signal. If this member is set, the driver ignores any bytes received, unless the DSR modem input line is high.

Bit 7: fTXContinueOnXoff: Specifies if transmission stops when the input buffer is full and the driver has transmitted the **sXOFF** character. If this member is set, transmission continues after the input buffer has come within **iXOFFLimit** bytes of being full and the driver has transmitted the **sXOFF** character to stop receiving bytes. If this member is cleared, transmission does not continue until the input buffer is within **iXONLimit** bytes of being empty and the driver has transmitted the **sXON** character to resume reception.

Bit 8: fOutX: Specifies if XON/XOFF flow control is used during transmission. If this member is set, transmission stops when the **sXOFF** character is received and starts again when the **sXON** character is received.

Bit 9: fInX: Specifies if XON/XOFF flow control is used during reception. If this member is set, the **sXOFF** character is sent when the input buffer comes within **iXOFFLimit** bytes of being full, and the **sXON** character is sent when the input buffer comes within **iXONLimit** bytes of being empty.

Bit 10: fErrorChar: Specifies if bytes received with parity errors are replaced with the character specified by the **sError** member. If this member is set and the **fParity** member is set, replacement occurs.

Bit 11: fNull: Specifies if null bytes are discarded. If this member is set, null bytes are discarded when received.

Bits 12-13: fRtsControl: Turns the RTS flow control on and off. RTS_CONTROL_ENABLE turns on the RTS line during the connection. RTS_CONTROL_HANDSHAKE turns on RTS handshaking. RTS_CONTROL_DISABLE turns off the RTS line. RTS_CONTROL_TOGGLE specifies that the RTS line is high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line will be low.

RTS_CONTROL_DISABLE: &h00
RTS_CONTROL_ENABLE: &h01
RTS_CONTROL_HANDSHAKE: &h02
RTS_CONTROL_TOGGLE: &h03

Bit 14: fAbortOnError: Specifies if read and write operations are terminated if an error occurs. If this member is set, the driver terminates all read and write operations with an error status if an error occurs.

Bits 15 – 31: Reserved. Must be cleared (0).

iXONLimit:

Optional parameter (default = 10). Specifies the minimum number of bytes accepted in the input buffer before the XON character is sent.

iXOFFLimit:

Optional parameter (default = 10). Specifies the maximum number of bytes accepted in the input buffer before the XOFF character is sent. The maximum number of bytes accepted is calculated by subtracting this value from the size, in bytes, of the input buffer.

iByteSize:

Optional parameter (default = 8). Specifies the bits per byte transmitted and received.

lParity:

Optional parameter (default = NO_PARITY). Specifies the parity scheme. Must be one of the following constants:

NO_PARITY: Parity checking not used.
ODD_PARITY: Sets the parity bit so that the count of bits set is an odd number.
EVEN_PARITY: Sets the parity bit so that the count of bits set is an even number.
MARK_PARITY: Leaves the parity bit set to 1.
SPACE_PARITY: Leaves the parity bit set to 0.

lStopBits:

Optional parameter (default = ONE_STOPBIT). Number of stop bits to be used. Stop bits separate each unit of data on an asynchronous serial connection. Must be one of the following constants:

ONE_STOPBIT: One stop bit.
ONE5_STOPBITS: One and a half stop bits.
TWO_STOPBITS: Two stop bits.

sXON:

Optional parameter (default = "", empty string). Value of the XON character for both transmission and reception. XON is a software control to resume the transmission of data (whereas RTS and CTS are hardware controls). XOFF stops the transmission. When the default empty string is passed the setting will not be changed from the current setting.

sXOFF:

Optional parameter (default = "", empty string). Value of the XOFF character for both transmission and reception. XOFF is a software control to stop the transmission of data (whereas RTS and CTS are hardware controls). XON resumes the transmission. When the default empty string is passed the setting will not be changed from the current setting.

sError;

Optional parameter (default = "", empty string). Value of the character used to replace bytes received with a parity error. When the default empty string is passed the setting will not be changed from the current setting.

sEOF:

Optional parameter (default = "", empty string). Value of the character used to signal the end of data. When the default empty string is passed the setting will not be changed from the current setting.

sEvent:

Optional parameter (default = "", empty string). Value of the control character that is used to signal an event, such as end of file. When the default empty string is passed the setting will not be changed from the current setting.

lInputBufSize:

Optional parameter (default = 1024). Specifies the recommended size, in bytes, of the device's internal input buffer.

lOutputBufSize:

Optional parameter (default = 512). Specifies the recommended size, in bytes, of the device's internal output buffer.

dReadIntervalTimeout:

Optional parameter (default = 0.1 seconds). Specifies the maximum acceptable time to elapse between the arrival of two characters on the communication line. The time period begins when the first character is received. If the interval between the arrival of any two characters exceeds this amount, the operation is completed and any buffered data is returned. A value of zero indicates that interval time-outs are not used.

dReadTotalTimeoutMult:

Optional parameter (default = 0.1 seconds). Specifies the multiplier used to calculate the total time-out period for read operations. For each read operation, this value is multiplied by the requested number of bytes to be read.

dReadTotalTimeoutConst:

Optional parameter (default = 0.1 seconds). Specifies the constant used to calculate the total time-out period for read operations. For each read operation, this value is added to the product of the **dReadTotalTimeoutMult** member and the requested number of bytes. A value of zero for both the **dReadTotalTimeoutMult** and **dReadTotalTimeoutConst** members indicates that total time-outs are not used for read operations.

dWriteTotalTimeoutMult:

Optional parameter (default = 0.1 seconds). Specifies the multiplier used to calculate the total time-out period for write operations. For each write operation, this value is multiplied by the number of bytes to be written.

dWriteTotalTimeoutConst:

Optional parameter (default = 0.1 seconds). Specifies the constant used to calculate the total time-out period for write operations. For each write operation, this value is added to the product of the **dWriteTotalTimeoutMult** member and the number of bytes to be written. A value of zero for both the **dWriteTotalTimeoutMult** and **dWriteTotalTimeoutConst** members indicates that total time-outs are not used for write operations.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

Use this function to control the exact configuration of the specified serial port.

Example:

Gather the response from a serial-based UUT running on COM2 at 4800 baud. Accept the default states of the remaining setup parameters:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE
sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\r\n"
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

See Also:

COMMInitialize, COMMProperties, SendReceive

19.50 SetLoad

Set the basic properties common to all loads.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **SetLoad (**
sLoadName As String,
[sValue As String = ""],
[lMode As LOAD_MODES = LOAD_NO_MODE],
[sVoltageRange As String = ""],
[sCurrentRange As String = ""],
[lPhaseControl As PHASE_CONTROL = NO_PHASE],
[hBlobAdv As Long = 0],
[bSendTrigger As Boolean = False]
) As Boolean

Parameters:

sLoadName:

The logical name of the load that will be set. This may be an actual device (e.g., "DC Load 1") or the name of an Output Container (e.g., "UUT Out 1").

sValue:

Optional parameter (default = "", empty string). This value is used to program the main setting of the load. The meaning of the value corresponds to the active load mode. It can be either a hard-coded value (e.g., "10.0"), or a percentage of a reference defined by **sLoadName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

lMode:

Optional parameter (default = LOAD_NO_MODE). This value is used to set the operating phase mode of the load. Must be one of the following constants:

LOAD_NO_MODE:	Do not change the active mode of the load.
LOAD_CONSTANT_CURRENT:	Place the load in Constant Current mode. **sValue** is Amps.
LOAD_CONSTANT_RESISTANCE:	Place the load in Constant Resistance mode. **sValue** is Ohms.
LOAD_SHORT_CIRCUIT:	Place the load in Short Circuit mode. **sValue** is ignored.
LOAD_CONSTANT_VOLTAGE:	Place the load in Constant Voltage mode. **sValue** is Volts.
LOAD_CONSTANT_POWER:	Place the load in Constant Power mode. **sValue** is Watts.

sVoltageRange:

Optional parameter (default = "", empty string). This value is used to set the voltage range of the load. It can be either a hard-coded value (e.g., "130.0"), or a percentage of the voltage reference defined by **sLoadName** (e.g., "120%"). If an empty string is passed the active value of this parameter will not be changed.

sCurrentRange:

Optional parameter (default = "", empty string). This value is used to set the current range of the load. It can be either a hard-coded value (e.g., "12.0"), or a percentage of the current reference defined by **sLoadName** (e.g., "120%"). If an empty string is passed the active value of this parameter will not be changed.

lPhaseControl:

Optional parameter (default = NO_PHASE). This value is used to set the operating phase mode of the load. This parameter is only used if **sLoadName** refers to an Output Container (not a device). Must be one of the following constants:

NO_PHASE:	Do not change the active phase mode of the load
ONE_PHASE:	Place the load in single-phase mode.
THREE_PHASE:	Place the load in three-phase mode.

hBlobAdv;

Optional parameter (default = 0). The handle to the blob defining the advanced properties of the load. This parameter is only used if *sLoadName* refers to a device (not an Output Container).

bSendTrigger:

If True, a system trigger will be sent to coincide with this operation.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

The parameters are sent to the load device or container in one package. The load determines the correct order in which to set the values. This function modifies only those optional parameters specified by the call (see Parameters). When the default value is used the corresponding property of the load remains unchanged from its current state.

Example:

Setup the output named “UUT Out 1” to a Constant Current setting of 100% of the reference current:

```
obTRHelper.SetLoad("UUT Out 1", "100%", LOAD_CONSTANT_CURRENT)
```

19.51 SetSource

Set the basic properties common to all sources.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **SetSource (**
sSourceName As String,
[sVoltage As String = ""],
[sVoltagePhaseB As String = ""],
[sVoltagePhaseC As String = ""],
[sCurrentLimit As String = ""],
[sCurrentLimitPhaseB As String = ""],
[sCurrentLimitPhaseC As String = ""],
[sVoltageRange As String = ""],
[sCurrentRange As String = ""],
[sFrequency As String = ""],
[lPhaseControl As PHASE_CONTROL = NO_PHASE],
[hBlobAdv As Long = 0],
[bSendTrigger As Boolean = False],
[sSettleDelay As String = ""]
) As Boolean

Parameters:

sSourceName:

The logical name of the source that will be set. This may be an actual device (e.g., "DC Source 1") or the name of an Input Container (e.g., "UUT In 1").

sVoltage:

Optional parameter (default = "", empty string). This value is used to set the voltage of the source. It can be either a hard-coded value (e.g., "120.0"), or a percentage of the voltage reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

sVoltagePhaseB:

Optional parameter (default = "", empty string). This value is used to set the phase B voltage of the source. It can be either a hard-coded value (e.g., "120.0"), or a percentage of the voltage reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

sVoltagePhaseC:

Optional parameter (default = "", empty string). This value is used to set the phase C voltage of the source. It can be either a hard-coded value (e.g., "120.0"), or a percentage of the voltage reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

sCurrentLimit:

Optional parameter (default = "", empty string). This value is used to set the current limit of the source. It can be either a hard-coded value (e.g., "10.0"), or a percentage of the current reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

sCurrentLimitPhaseB:

Optional parameter (default = "", empty string). This value is used to set the phase B current limit of the source. It can be either a hard-coded value (e.g., "10.0"), or a percentage of the current reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

sCurrentLimitPhaseC:

Optional parameter (default = "", empty string). This value is used to set the phase C current limit of the source. It can be either a hard-coded value (e.g., "10.0"), or a percentage of the current reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

sVoltageRange:

Optional parameter (default = "", empty string). This value is used to set the voltage range of the source. It can be either a hard-coded value (e.g., "130.0"), or a percentage of the voltage reference defined by **sSourceName** (e.g., "120%"). If an empty string is passed the active value of this parameter will not be changed.

sCurrentRange:

Optional parameter (default = "", empty string). This value is used to set the current range of the source. It can be either a hard-coded value (e.g., "12.0"), or a percentage of the current reference defined by **sSourceName** (e.g., "120%"). If an empty string is passed the active value of this parameter will not be changed.

sFrequency:

Optional parameter (default = "", empty string). This value is used to set the frequency of the source. It can be either a hard-coded value (e.g., "60.0"), or a percentage of the frequency reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed the active value of this parameter will not be changed.

lPhaseControl:

Optional parameter (default = NO_PHASE). This value is used to set the operating phase mode of the source. Must be one of the following constants:

NO_PHASE:	Do not change the active phase mode of the source
ONE_PHASE:	Place the source in single-phase mode. **sVoltagePhaseB**, **sVoltagePhaseC sCurrentLimitPhaseB** and **sCurrentLimitPhaseC** are ignored.
THREE_PHASE:	Place the source in three-phase mode, interpret **sVoltage** as a line-to-line value applied to all phases and set the current limit of all phases to the value in **sCurrentLimit**. **sVoltagePhaseB**, **sVoltagePhaseC sCurrentLimitPhaseB** and sCurrentLimitPhaseC are ignored.
THREE_PHASE_COMPLEX:	Place the source in three-phase mode and interpret the voltage values as individual line-to-neutral values.

hBlobAdv:

Optional parameter (default = 0). The handle to the blob defining the advanced properties of the source. This parameter is only used if **sSourceName** refers to a device (not an Input Container).

BSendTrigger:

Optional parameter (default = False). If True, a system trigger will be sent to coincide with this operation.

sSettleDelay:

Optional parameter (default = "", empty string). This value is simply a delay that is performed after the setup defined by this function has been applied. It can be either a hard-coded value (e.g., "100 mS"), or a percentage of the time reference defined by **sSourceName** (e.g., "100%"). If an empty string is passed no delay is performed.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

The parameters are sent to the source device or container in one package. The source determines the correct order in which to set the values. This function modifies only those optional parameters specified by the call (see Parameters). When the default value is used the corresponding property of the source remains unchanged from its current state.

Example:

Setup the input named "UUT In 1" to a voltage value of 100% of reference voltage and a current limit = 110% of reference current:

```
obTRHelper.SetSource("UUT In 1", "100%", "", "", "110%")
```

19.52 SetVector

Apply the named Input, Output, Digital or Analog vector.
Member of NHRINTERFACESLib.INHRTRHelper

```
Sub SetVector (
                sVectorName As String,
                [bSendTrigger As Boolean = False]
              )
```

Parameters:

sVectorName:

The name of the vector that will be set.

bSendTrigger:

Optional parameter (default = False). If True, a system trigger will be sent to coincide with this operation.

Return Values:

None

Remarks:

This function will apply the named Input, Output, Digital or Analog vector to tester hardware. If **bSendTrigger** is True, a system trigger will be sent to coincide with the operation. A fault will be generated if the tester fails to generate the trigger when requested. The most common reason a trigger is not generated is because the tester hardware is already at the state defined by the named vector – in this case, no commands are sent to hardware.

If **sVectorName** is an empty string and **bSendTrigger** is True, the function will generate a system trigger without changing the state of the tester hardware.

Example #1:

Apply the vector named “High Line” without generating a system trigger:

```
obTRHelper.SetVector "High Line"
```

Example #2:

Apply the vector named “High Line” and issue a system trigger:

```
obTRHelper.SetVector "High Line", True
```

Example #3:

Generate a system trigger without changing the state of the tester hardware:

```
obTRHelper.SetVector "", True
```

See Also:

GetAnalogVectorName, GetDigitalVectorName, GetInVectorName, GetOutVectorName

19.53 StrResult

Log a Result with string data.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **StrResult (**
sValue As String,
lResultType As RESULT_TYPE,
[sLabel As String = ""],
[sMinimum As String = ""],
[sMaximum As String = ""],
[sResult As String = ""],
[lIndent As Long = 0],
[lParentGroup As Long = 0],
[lSortOrder As Long = 0]
) As Long

Parameters:

sValue:

The string value being logged. This string is placed in the "Actual" field of the datalog device.

lResultType:

The results of the measurement evaluation. This must be one of the RESULT_TYPE enumeration values. Datalog drivers use this value when determining the pass or fail state of this measurement.

sLabel:

A string defining the value being logged. This value is placed in the "Label" field of the datalog device. May be an empty string if no label is desired.

sMinimum:

The minimum acceptable string for the measurement being logged. This string is placed in the "Minimum" field of the datalog device. May be an empty string if no minimum string is desired

sMaximum:

The maximum acceptable string for the measurement being logged. This string is placed in the "Maximum" field of the datalog device. May be an empty string if no maximum string is desired.

sResult:

A string representation of the results of the measurement evaluation, usually either "Pass", "Fail" or "Untested". This string is placed in the "Result" field.

lIndent:

Optional parameter (default = 0). The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

lParentGroup:

Optional parameter (default = 0). Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Optional parameter (default = 0). Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

Return Values

Returns a new Parent Group index that can be used as the **lParentGroup** parameter in subsequent datalog calls.

Remarks:

This command logs the supplied data strings to all the configured dataloggers. The drivers use the **lResultType** parameter to determine the pass/fail state of the measurement, and not the

sResult parameter. The result may or may not be written to the datalog device depending on the configuration of the datalog driver.

Within the same test routine, multiple values logged with the same lParentGroup number are grouped together and sub-sorted according to **lSortOrder**. In the case of duplicate **lSortOrder** values, the values are logged in the order they were received. You may pass zero values for both sorting parameters to log all measurement in the order they were received. **lParentGroup** and **lSortOrder** parameters are not maintained across separate test routine executions. Refer to the example under "ValResult".

Example:

Send a string to the dataloggers:

```
Dim sMeasurement As String

sMeasurement = "123ABC"
obTRHelper.StrResult sMeasurement, FAIL_RESULT, "UUT Communications", "ABC123",
"", "Fail"
```

Sample Screen driver output:

Label	Minimum	Actual	Maximum	Result
UUT Communications	ABC123	123ABC		Fail

See Also:

EvaluateStrResult, ValResult

19.54 TimingMeas

Perform a timing measurement cycle.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **TimingMeas (**
sMeasVectorName As String,
[sVectorName As String = “”],
[sStartEventTimeout As String = “10.0 S”]
) As Double

Parameters:

sMeasVectorName:

The name of the Measurement Vector defining the desired measurement.

sVectorName:

Optional parameter (default = “”, empty string). An Input, Output, Analog or Digital Vector which will be set after the instrument is armed but before the measurement is taken. A system trigger will be sent to coincide with this operation.

sStartEventTimeout:

Optional parameter (default = 10 seconds). The amount of time (in seconds) to wait for the trigger before aborting the measurement cycle.

Return Values:

Returns the value measured by the instrument.

Remarks:

This function will set up the instrument to take the measurement defined in the Measurement Vector. If defined, the Vector named **sVectorName** is applied after the instrument is armed.

Example:

Perform the timing measurement defined by the Measurement Vector named “Rise Time”. The function will apply the vector named “Nom Line” and abort if a trigger is not detected within 10 seconds:

```
Dim dMeasurement As Double
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lFailCount As Long

dMeasurement = obTRHelper.TimingMeas("Rise Time", "Nom Line")
lResult = obTRHelper.EvaluateValResult("50.0E-6", dMeasurement, "75.0E-6",
sResult)
obTRHelper.ValResult dMeasurement, lResult, "Rise Time", "50.0E-6", "75.0E-6",
"S", sResult
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

EvaluateValResult, ValResult

19.55 Tokenize

Break the returned UUT communications string into tokens.
Member of NHRINTERFACESLib.INHRTRHelper

Sub Tokenize (
sPortName As String,
bEnable As Boolean,
[vDelimiter As Variant],
[lTokenNumber As Long = 1],
[bIgnoreRepeats As Boolean = True]
)

Parameters:

sPortName:

The logical name of the UUT Communications object on which to perform this operation. This must be an entry currently configured in the *em**Power***™ program.

bEnable:

True to tokenize the UUT response according to the delimiter value, **False** to leave the response string intact.

vDelimiter:

Optional parameter. Either a string containing the data pattern to match, or a numeric value indicating how many characters each token should contain.

lTokenNumber:

Optional parameter (default = 1). The token to use as the UUT response – all other data is removed from the response string. Valid numbers are 1 to the number of tokens detected.

bIgnoreRepeats:

Optional parameter (default = True). Reserved for future use.

Return Values:

None

Remarks:

Use this function to define the tokenization actions that will be performed on the response string during a **SendReceive** operation.

Example:

Tokenize the received data using a “space” character as the delimiter, and consider the third token to be the UUT response:

```
Dim sPort As String
Dim sResponse As String
Dim sResult As String
Dim sDescription As String
Dim lResultType As RESULT_TYPE

sPort = "UUT Communications"
SerialSetup sPort, 2, 4800
COMMInitialize sPort
COMMProperties sPort, False, False, "", True, "\r\d"
Tokenize sPort, True, " ", 3
SendReceive sPort, sResponse, sResult, sDescription, lResultType
```

If the UUT returned “A response from the UUT”, five tokens would be detected. The selected token would be “from”.

See Also:

SerialSetup, COMMInitialize, COMMProperties, SendReceive

19.56 TransientMeas

Perform a transient measurement cycle.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **TransientMeas (**
lTransientFunction As TRANSIENT_FUNCTION,
sDeviceName As String,
sVectorName As String,
sMeasVectorName As String,
sChannelName As String,
[sRange As String = "0.0 V"],
[sDelay As String = "0.0 S"],
[sMinVoltage As String = "0.0 V"],
[sMaxVoltage As String = "0.0 V"]
) As Double

Parameters:

lTransientFunction:

The type of transient function to perform. Must be one of the following constants:

TRANSIENT_UNDERSHOOT:	An AC coupled undershoot measurement.
TRANSIENT_OVERSHOOT:	An AC coupled overshoot measurement.
TRANSIENT_SETTLE_VOLTAGE:	A DC coupled voltage measurement taken after the aperture.
TRANSIENT_SETTLE_TIME:	A DC coupled time-to-settle measurement
TRANSIENT_DC_UNDERSHOOT:	A DC coupled undershoot measurement.
TRANSIENT_DC_OVERSHOOT:	A DC coupled overshoot measurement.

sDeviceName:

The logical name of the device that will be set by the vector generating the transient condition. This may be the name of an Input, Output or Digital Container or Analog device (e.g., "UUT Out 1").

sVectorName:

An Input, Output, Digital or Analog Vector that will be used to generate the transient. In order for the transient to occur, this vector must modify the state of the device defined by **sDeviceName**. A system trigger will be sent to coincide with this operation.

sMeasVectorName:

The name of the Measurement Vector defining the desired transient.

sChannelName:

The name of the channel to be measured. This may be either an actual multiplexer channel (e.g., "Chn 001") or the name of an Input or Output Container (e.g., "UUT Out 1").

sRange:

Optional parameter (default = 0 volts). This value is used to set the measurement range of the instrument. It can be either a hard-coded value (e.g., "5.5"), or a percentage of the voltage reference defined by **sChannelName** (e.g., "110%"). Good practice is to select a maximum value that you expect will never, under normal circumstances, be exceeded by the measurement. This will place the instrument is the range most appropriate to the measurement. A value of zero is interpreted as the maximum range capability of the measurement instrument.

sDelay:

Optional parameter (default = 0 seconds). The amount of time (in seconds) to delay before applying the transient. This value can be either a hard-coded value (e.g., "100 mS"), or a percentage of the time reference defined by **sChannelName** (e.g., "110%").

sMinVoltage:

Ignored parameter unless **lTransientFunction** is TRANSIENT_SETTLE_TIME (default = 0 volts). The minimum voltage where the transient is considered to be settled. "Settled" is when the signal remains within the boundary defined by **sMinVoltage** and **sMaxVoltage**. It can be either a hard-coded value (e.g., "5.5"), or a percentage of the voltage reference defined by **sChannelName** (e.g., "110%").

sMaxVoltage:

Ignored parameter unless **lTransientFunction** is TRANSIENT_SETTLE_TIME (default = 0 volts). The maximum voltage where the transient is considered to be settled. "Settled" is when the signal remains within the boundary defined by **sMinVoltage** and **sMaxVoltage**. It can be either a hard-coded value (e.g., "5.5"), or a percentage of the voltage reference defined by **sChannelName** (e.g., "110%").

Return Values:

Returns the value measured by the instrument.

Remarks:

This function will set up the instrument to take the transient measurement defined by the **lTransientFunction** parameter. The Vector named **sVectorName** is applied to the device named **sDeviceName** after the instrument is armed.

Example:

Perform an undershoot test on the output named "UUT Out 1" by applying the vector named "Max Load" to that output:

```
Dim dMeasurement As Double
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lFailCount As Long

dMeasurement = obTRHelper.TransientMeas(TRANSIENT_UNDERSHOOT, "UUT Out 1", "Max
Load", "Transient","UUT Out 1")
lResult = obTRHelper.EvaluateValResult("0.01", dMeasurement, "0.1", sResult)
obTRHelper.ValResult dMeasurement, lResult, "Volts", "0.01", "0.1", "V", sResult
If lResult = FAIL_RESULT
    lFailCount = lFailCount + 1
End If
```

See Also:

EvaluateValResult, ValResult

19.57 ValResult

Log a Result with numeric data.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **ValResult (**
dValue As Double,
lResultType As RESULT_TYPE,
[sLabel As String = ""],
[sMinimum As String = ""],
[sMaximum As String = ""],
[sUnits As String = ""],
[sResult As String = ""],
[lIndent As Long = 0],
[lParentGroup As Long = 0],
[lSortOrder As Long = 0]
) As Long

Parameters:

dValue:

The numeric value being logged. This value is placed in the "Actual" field of the datalog device.

lResultType:

The results of the measurement evaluation. This must be one of the RESULT_TYPE enumeration values. Datalog drivers use this value when determining the pass or fail state of this measurement.

sLabel:

Optional parameter (default = "", empty string). A string defining the value being logged. This value is placed in the "Label" field of the datalog device. May be an empty string if no label is desired.

sMinimum:

Optional parameter (default = "", empty string). The minimum acceptable value for the measurement being logged. This value is placed in the "Minimum" field of the datalog device.

sMaximum:

Optional parameter (default = "", empty string). The maximum acceptable value for the measurement being logged. This value is placed in the "Maximum" field of the datalog device.

sUnits:

Optional parameter (default = "", empty string). The units of the value being logged (e.g., "V", "Hz", etc.) May be an empty string if no units are desired.

sResult:

Optional parameter (default = "", empty string). A string representation of the results of the measurement evaluation, usually either "Pass", "Fail" or "Untested". This string is placed in the "Result" field.

lIndent:

Optional parameter (default = 0). The indent level for the provided label. An indent level is equivalent to one tab stop of the datalog device.

lParentGroup:

Optional parameter (default = 0). Used for sorting the logged values. This is the parent index used to control the placement of this entry (see Remarks).

lSortOrder:

Optional parameter (default = 0). Used for sorting the logged values. This is the sorting parameter within a parent group (see Remarks).

Return Values:

Returns a new Parent Group index that can be used as the **lParentGroup** parameter in subsequent datalog calls.

Remarks:

This command logs the supplied data values to all the configured dataloggers. The drivers use the **lResultType** parameter to determine the pass/fail state of the measurement, and not the **sResult** parameter. The result may or may not be written to the datalog device depending on the configuration of the datalog driver.

Within the same test routine, multiple values logged with the same **lParentGroup** number are grouped together and sub-sorted according to **lSortOrder**. In the case of duplicate **lSortOrder** values, the values are logged in the order they were received. You may pass zero values for both sorting parameters to log all measurement in the order they were received. **lParentGroup** and **lSortOrder** parameters are not maintained across separate test routine executions.

Example:

The following example illustrates the effect of **lParentGroup** and **lSortOrder**. Assume that the first measurement cycle gathers voltage measurements and the second gathers current measurements but you desire the voltage and current measurements for each channel to be logged next to each other.

Since the **lParentGroup** and **lSortOrder** values of the three grouping labels are zero, they become root entries, sorted in the order they are datalogged (see Sample Screen driver output). The "Volts" and "Amps" values are logged with the return value of these grouping labels as the **lParentGroup** values, making them children of the corresponding label. Since the "Volts" **lSortOrder** value is less than the "Amps" sort order, "Volts" appears first in the datalog:

```
Dim dMeasurement(0 To 2) As Double
Dim i As Integer
Dim lResult As RESULT_TYPE
Dim sResult As String
Dim lGroup(0 To 2) As Long

lGroup(0) = obTRHelper.Label("Channel 1")
lGroup(1) = obTRHelper.Label("Channel 2")
lGroup(2) = obTRHelper.Label("Channel 3")

dMeasurement(0) = obTRHelper.DMMMeas("DMM DCV", "Chn 001", "5.5")
dMeasurement(1) = obTRHelper.DMMMeas("DMM DCV", "Chn 002", "5.5")
dMeasurement(2) = obTRHelper.DMMMeas("DMM DCV", "Chn 003", "5.5")

For i = 0 To 2
   lResult = obTRHelper.EvaluateValResult("4.95", dMeasurement(i), "5.05",
sResult)
   obTRHelper.ValResult dMeasurement(i), lResult, "Volts", "4.95", "5.05", "V",
sResult, 1, lGroup(i), 0
Next

dMeasurement(0) = obTRHelper.DMMMeas("DMM DCA", "Chn 001", "3.0")
dMeasurement(1) = obTRHelper.DMMMeas("DMM DCA", "Chn 002", "3.0")
dMeasurement(2) = obTRHelper.DMMMeas("DMM DCA", "Chn 003", "3.0")

For i = 0 To 2
   lResult = obTRHelper.EvaluateValResult("1.1", dMeasurement(i), "2.5", sResult)
   obTRHelper.ValResult dMeasurement(i), lResult, "Amps", "1.1", "2.5", "A",
sResult, 1, lGroup(i), 1
Next
```

Sample Screen driver output:

Label	Minimum	Actual	Maximum	Result
Channel 1				
Volts	4.950 V	4.997 V	5.050 V	Pass
Amps	1.100 A	1.736 A	2.500 A	Pass
Channel 2				
Volts	4.950 V	5.025 V	5.050 V	Pass
Amps	1.100 A	1.583 A	2.500 A	Pass

```
Channel 3
    Volts          4.950 V      4.967 V      5.050 V      Pass
    Amps           1.100 A      2.024 A      2.500 A      Pass
```

See Also:

Label, DMMMeas, EvaluateValResult

19.58 WaveformCapture

Capture a waveform.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **WaveformCapture (**
sWaveformDevName As String,
hBlob As Long,
lNumChannels As Long,
lNumPoints As Long,
dSampleRate As Double,
dTriggerTime As Double,
[sVectorName As String = “”],
[sStartEventTimeout As String = “10.0 S”],
= “0.0 S”]
) As Long

Parameters:

sWaveformDevName:

The logical name of the waveform device capturing the data.

hBlob:

The handle to the blob defining the desired measurement waveform. May be zero if the measurement definition was passed with **WaveformSetup**.

lNumChannels:

This variable will be set to the number of channels captured in the waveform.

lNumPoints:

This variable will be set to the number of points collected per channel.

dSampleRate:

This variable will be set to the sample rate of the captured waveform.

dTriggerTime:

This variable will be set to the time that the trigger occurred within the waveform.

sVectorName:

Optional parameter (default = “”, empty string). An Input, Output, Analog or Digital Vector which will be set after the instrument is armed but before the waveform is captured. A system trigger will be sent to coincide with this operation.

sStartEventTimeout:

Optional parameter (default = 10 seconds). The amount of time (in seconds) to wait for the trigger before aborting the measurement cycle.

sDelay:

Optional parameter (default = 0 seconds). The amount of time (in seconds) to delay before the measurement.

Return Values:

Returns an ownership handle. A non-zero value indicates success in obtaining ownership of the waveform device and capturing the desired waveform; a zero indicates a failure to obtain ownership or capture the waveform. This ownership handle must be “returned” to the system through a **WaveformRelease** call.

Remarks:

This function will set up the instrument to capture the waveform defined in the Measurement Vector. If defined, the Vector named **sVectorName** is applied after the instrument is armed. The waveform is captured after the programmed delay. If the waveform has been successfully captured, the **lNumChannels**, **lNumPoints, dSampleRate** and **dTriggerTime** variables are set to contain data that describes the waveform.

Example:

Capture a waveform defined by the Measurement Vector “Waveform 1”:

```
lOwnershipLock = obTRHelper.WaveformCapture("Waveform 1", hblob, lNumChannels,
lNumPoints, dSampleRate, dTriggerTime)

'Transfer waveform data here (see WaveformData)

obTRHelper.WaveformRelease("Waveform 1", lOwnershipLock)
```

See Also:

WaveformData, WaveformRelease

19.59 WaveformData

Get the data points from the captured waveform.
Member of NHRINTERFACESLib.INHRTRHelper

Function: **WaveformData (**
sWaveformDevName As String,
lChannel As Long,
arydPoints() As Variant,
sChannel As String,
sUnits As String,
dScaleMin As Double,
dScaleMax As Double
) As Boolean

Parameters:

sWaveformDevName:

The logical name of the waveform device capturing the data.

lChannel:

Transfer waveform data from this waveform channel (see Remarks).

arydPoints:

An array of doubles containing enough elements to hold the waveform data for the indicated channel. This parameter is passed as a Variant.

sChannel:

This variable will be set to the description of the selected waveform channel.

sUnits:

This variable will be set to the measurement unit description of the selected channel.

dScaleMin:

This variable will be set to the minimum range value encompassing the captured data.

dScaleMax:

This variable will be set to the maximum range value encompassing the captured data.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

This function will transfer the waveform data for the indicated channel into a user-supplied array of doubles. In addition, several variables will be set to values describing the data (see Parameters).

Normally, this function is called once for each channel captured as indicated by **WaveformCapture**::**lNumChannels**. Note that **lChannel** is zero-based (0 to **lNumChannels** – 1).

Example:

Capture a waveform defined by the Measurement Vector “Waveform 1”:

```
ReDim dPoints(0 To lNumPoints – 1)
For lChannel = 0 To lNumChannels - 1
    lCount = obTRHelper.WaveformData("Waveform 1", lChannel, dPoints, sChannel,
sUnits,
                                     dScaleMin, dScaleMax)
    'Do something with the data...
Next
```

See Also:

WaveformCapture, WaveformRelease

19.60 WaveformRelease

Release the lock on a waveform device.
Member of NHRINTERFACESLib.INHRTRHelper

Sub WaveformRelease (
sWaveformDevName As String,
lOwnershipLock As Long
)

Parameters:

sWaveformDevName:

The logical name of the waveform device capturing the data.

lOwnershipLock:

The ownership lock value returned from a **WaveformCapture** function call.

Return Values:

None

Remarks:

This function will release a lock placed on a waveform device during a waveform capture. The lock must be placed on the device in order to keep other threads and processes from corrupting the waveform data before the user has copied it into their own local storage.

Example:

Unlock the waveform device previously locked via WaveformCapture:

```
obTRHelper.WaveformSetup("Waveform 1", 0.1, lChannel, sUnits, sMuxChannel)
lOwnershipLock = obTRHelper.WaveformCapture("Waveform 1", 0, lNumChannels,
lNumPoints, dSampleRate, dTriggerTime)

'Transfer waveform data here (see WaveformData)

obTRHelper.WaveformRelease("Waveform 1", lOwnershipLock)
```

See Also:

WaveformSetup, WaveformCapture, WaveformData

19.61 WaveformSetup

Setup a device to capture a waveform.
Member of NHRINTERFACESLib.INHRTRHelper

```
Function :  WaveformSetup (
                sWaveformDevName As String,
                dAperture As Double,
                arylChannel() As Variant,
                arysUnits() As Variant,
                arysMuxChannelName() As Variant,
                [lArraySize As Long = 1],
                [dSampleRate As Double = 0.0],
                [lAcquisitionType As WAVE_FORM_ACQUISITION_TYPE = NORMAL],
                [dTriggerPosition As Double = 0.0],
                [dTriggerLevel As Double = 0.0],
                [lTriggerCoupling As TRIGGER_COUPLING = DC_COUPLED_TRIGGER],
                [lTriggerSlope As TRIGGER_SLOPE = RISING],
                [lTriggerSource As TRIGGER_SOURCE = SYSTEM_TRIGGER],
                [lTriggerChannel As Long = 0],
                [sTriggerName As String = ""],
                [lTriggerType As TRIGGER_TYPE = IMMEDIATE],
                [lTriggerMode As TRIGGER_MODE = AUTO_MODE],
                [arybEnable() As Variant = True],
                [arydRange() As Variant = 0.0],
                [arydOffset() As Variant = 0.0],
                [arydScaleFactor() As Variant = 1.0],
                [arylBandWidthFilter() As WAVE_FORM_BAND_WIDTH_FILTER =
                BAND_WIDTH_FULL],
                [arylSignalCoupling() As SIGNAL_COUPLING = DC_COUPLED],
                [arylResolutionBits() As Long = 0],
                [hBlobAdv As Long = 0]
            ) As Boolean
```

Parameters:

sWaveformDevName:

The logical name of the waveform device capturing the data.

dAperture:

The measurement window aperture size, in seconds.

arylChannel:

An array of longs defining the desired waveform channels to use. This parameter is passed as a Variant.

arysUnits:

An array of string defining the measurement units for each channel. Each channel may have it's own measurement unit. This parameter is passed as a Variant.

arysMuxChannelName:

An array of strings defining the multiplexer channels used by each channel. This parameter is passed as a Variant.

lArraySize:

Optional parameter (default = 1). The number of channels defined by the array parameters.

dSampleRate:

Optional parameter (default = 0.0). The number of samples digitized per second. The default value instructs the hardware to sample as fast as possible.

lAcquisitionType:

Optional parameter (default = NORMAL). The speed of the data acquisition. Must be one of the following constants:

NORMAL:	Perform the highest capture rate by sampling as often as possible
FAST:	Perform a quick capture with minimal data by sampling less often

dTriggerPosition:

Optional parameter (default = 0.0 seconds). The data position within the aperture where the trigger is to be placed.

dTriggerLevel:

Optional parameter (default = 0.0). The voltage level of the trigger signal at which the measurement device will begin capturing data.

lTriggerCoupling:

Optional parameter (default = DC_COUPLED_TRIGGER). The coupling of the trigger signal. Must be one of the following constants:

DC_COUPLED_TRIGGER:	The trigger signal is directly coupled to the measurement circuitry
AC_COUPLED_TRIGGER:	The trigger signal is coupled through a capacitor to remove the signal's DC component

lTriggerSlope:

Optional parameter (default = RISING). The direction the trigger signal is traveling. Must be one of the following constants:

RISING	The signal is moving from a less positive to a more positive value
FALLING:	The signal is moving from a more positive to a less positive value

lTriggerSource:

Optional parameter (default = SYSTEM_TRIGGER). The signal source of the trigger. Must be one of the following constants:

CHANNEL:	The channel number specified by an entry in **arylChannel**. Use **lTriggerChannel** to specify the array entry to use.
AC_PRESENT:	The AC Present signal of the test system
DC_PRESENT:	The DC Present signal of the test system
SYSTEM_TRIGGER:	The standard system trigger
EXTERNAL_TRIGGER:	An external signal
DIGITAL:	The digital input specified by **sTriggerName**

lTriggerChannel:

Optional parameter (default = 0). The zero-based array entry in arylChannel to use as the trigger source.

sTriggerName:

Optional parameter (default = "", empty string). The name of the signal used for triggering. This entry is used when the lTriggerSource selection is DIGITAL.

lTriggerType:

Optional parameter (default = IMMEDIATE). Defines the trigger method used when the **WaveformCapture** command is sent. Must be one of the following constants:

IMMEDIATE:	Begin sampling the data without waiting for a trigger
EDGE:	Wait for the trigger to occur before sampling data

lTriggerMode:

Optional parameter (default = AUTO_MODE). Must be one of the following constants:

AUTO_MODE:	Auto-trigger is the trigger signal does not occur before the timeout period
NORMAL_MODE:	Wait infinitely for the trigger signal to occur.

arybEnable:

Optional parameter (default = True). An array of BOOLs (long integers) defining whether the corresponding waveform channel is enabled. Each array element corresponds to an entry in the arylChannel array. This parameter is passed as a Variant.

arydRange:

Optional parameter (default = 0.0). This value is used to set the measurement range of the corresponding channel. Each array element corresponds to an entry in the arylChannel array. Good practice is to select a maximum value that you expect will never, under normal circumstances, be exceeded by the measurement. This will place the instrument is the range most appropriate to the measurement. A value of zero is interpreted as the maximum channel range capability of the measurement instrument. This parameter is passed as a Variant.

arydOffset:

Optional parameter (default = 0.0). An offset value added to each data point in the captured data. Each array element corresponds to an entry in the **arylChannel** array. Use this feature to adjust the captured data for any DC bias before returning the data to the user. This parameter is passed as a Variant.

arydScaleFactor:

Optional parameter (default = 1.0). A scaling value multiplied with each data point in the captured data. Each array element corresponds to an entry in the **arylChannel** array. Use this feature to compensate for a voltage divider network or to translate the voltage measured from a current shunt into an amperage value. This parameter is passed as a Variant.

arylBandWidthFilter:

Optional parameter (default = BAND_WIDTH_FULL). Reserved for future use. Must be BAND_WIDTH_FULL. Each array element corresponds to an entry in the **arylChannel** array. This parameter is passed as a Variant.

arylSignalCoupling:

Optional parameter (default = DC_COUPLED). The coupling of the signal. Each array element corresponds to an entry in the arylChannel array. This parameter is passed as a Variant. Must be one of the following constants:

DC_COUPLED: The signal is directly coupled to the measurement circuitry
AC_COUPLED: The signal is coupled through a capacitor to remove the signal's DC component
GROUNDED: The measurement circuitry is connected to an internal ground

arylResolutionBits:

Optional parameter (default = 0). The analog-to-digital conversion resolution of the measurement. Higher resolution values generate more accurate conversions. Lower resolution values perform a faster conversion. The default of zero instructs the measurement to perform at its highest conversion resolution. Each array element corresponds to an entry in the **arylChannel** array. This parameter is passed as a Variant.

hBlobAdv:

Optional parameter (default = 0). The handle to the blob defining the advanced properties of the waveform measurement.

Return Values:

Returns **True** if successful; **False** if errors occurred.

Remarks:

This function will provide the instrument with the setup necessary to capture the desired waveform.

Example:

Capture a waveform:

```
obTRHelper.WaveformSetup("Waveform 1", 0.1, lChannel, sUnits, sMuxChannel)
lOwnershipLock = obTRHelper.WaveformCapture("Waveform 1", 0, lNumChannels,
lNumPoints, dSampleRate, dTriggerTime)

'Transfer waveform data here (see WaveformData)

obTRHelper.WaveformRelease("Waveform 1", lOwnershipLock)
```

See Also:

WaveformCapture, WaveformData, WaveformRelease