Proprietary Notice

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1 Purpose 1
1.2 Scope 1
1.3 Applicable Documents 1
1.4 Codes and Standards 2
1.5 Abbreviations and Definitions 2

2. DESIGN 3

2.1 Introduction 3
2.2 Collection Interface 4
2.3 Interface "Errors" 6
2.4 Interface "TestResult" 8
2.5 Interface "TestInfo" 12
2.6 Interface "Fullset" 19
2.7 Interface "Channel" 29
2.8 Interface "SensorCal" 35
2.9 Interface "Polynomial" 42
2.10 Interface "BPTable2D" 44
2.11 Interface "BPTable3D" 46
2.12 Interface "EngineeringUnits" 49

1. INTRODUCTION

1.1 Purpose

1.1.1 proDAS (professional Data Acquisition System) is a data acquisition system for gas turbine test cells. This document defines the interface for the Windows Database Server.

1.1.2 The Database Server provides the interface between the TRSCDB and the applications running in the Windows environment (e.g. Fullset Recalculation, User Functions and Text Output Pages). The components access the data via a COM interface, i.e. the Database Server is a COM server.

1.1.3 The TRSCDB stores the data related to Test Results and Sensor Calibration.

1.2 Scope

1.2.1 This document is intended for programmers of the COM client components using the COM interface(s) specified herein as well as the programmers of the server program offering the COM interface(s).

1.3 Applicable Documents

Number	Title
ES78001.2620	Functional Requirements Document for proDAS
ES78031.2664	Engineering Specification for Windows Database Server
ES78024.2673	Interface Control Document for the User Security System
DB78031.2702	Design Brief for Windows Database Server
ER78011.2573	Engineering Report for Test Configuration and Sensor Calibration Database

1.4 Codes and Standards

Number	Title
N/A	COM Interface Control Document Template

1.5 Abbreviations and Definitions

Term	Definition
May	An option or permission
Shall	A mandatory requirement
Should	A recommendation
Will	A statement of intent
COM	Component Object Model
DCOM	Distributed Component Object Model
EXE	File extension for executable files
ICD	Interface Control Document
IDL	Interface Definition Language
MDS	MDS Aero Support Corporation
MTU	MTU Aero Engines
ODBC	Open Database Connectivity
proDAS	professional Data Acquisition System
TRSCDB	Test Result and Sensor Calibration Database
WDBS	Windows Database Server

2. DESIGN

2.1 Introduction

2.1.1 The server shall be an out-of-process server.

2.1.2 The server shall be an executable, i.e. it shall have the file extension .exe.

2.1.3 There shall only be one instance of the server running within proDAS.

2.1.4 The server shall be thread-safe, i.e. several clients shall be able to access it simultaneously.

2.1.5 The server shall be self-registering by calling it with the argument "/RegServer". Registration will take place when proDAS is installed.

2.1.6 The server shall have a configurable time-out read at initialisation time from an installation dependant configuration file, i.e. if the time is set to 15 seconds; it shall terminate 15 seconds after the last connection has been destroyed. The default time-out value shall be set to 5 seconds.

2.1.7 For single item access the appropriate types shall be used (instead of variables without types).

2.1.8 Access of single item data shall be achieved by the DCOM interfaces TestResult (cf. 2.4) and SensorCal (cf. 2.8).

2.1.9 There will be no standard interfaces implemented other than IUnknown, which is implemented with every COM interface.

2.2 Collection Interface

2.2.1 Description

2.2.1.1 This interface will be used for all collections. It is the standard method to implement automation collections.

2.2.2 Design

2.2.2.1 This interface shall be a dispatch interface.

2.2.2.2 This interface shall be an automation interface.

2.2.3 Methods and Properties

2.2.3.1 Property Count

```
// Number of objects
[propget, id(1)]
HRESULT Count([out, retval] long *pVal);
```

Argument Name	Description
*pVal	number of objects in the collection.

2.2.3.2 Property _NewEnum

```
// Accessing the enumeration interface
[propget, id(DISPID_NEWENUM)]
      HRESULT _NewEnum([out, retval] LPUNKNOWN *pVal);
```

Argument Name	Description
*pVal	an interface *IEnumVARIANT*, used for enumerating the members of the collection.

2.2.3.3 Property Item

```
// Accessing an item by index
[propget, id(DISPID_VALUE)]
HRESULT Item([in] long Index, [out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
*pVal	the item with index *Index*, where indexing shall start with 1. If *Index* is not between 1 and *Count* an error *E_INVALIDARG* shall be generated.

2.3 Interface "Errors"

2.3.1 Description

2.3.1.1 This interface represents the buffer for the error messages generated when checking configuration data.

2.3.2 Design

2.3.2.1 This interface shall be a dispatch interface.

2.3.2.2 This interface shall be an automation interface.

2.3.2.3 This interface shall not be directly created. It shall be created by the *Errors* property of the interface *TestResult* or *SensorCal*.

2.3.2.4 Note that this interface will **not** implement a collection interface.

2.3.2.5 All properties shall be local.

2.3.3 Methods and Properties

2.3.3.1 Property IsEmpty

```
[propget, id(1)]
HRESULT IsEmpty([out, retval] BOOL *pVal);
```

Argument Name	Description
*pVal	return *true*, if the message buffer is empty and *false* otherwise.

2.3.3.2 Property Message

```
[propget, id(2)]
HRESULT Message([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the oldest message, if there is one, otherwise an empty string. The returned message shall be removed from the buffer.

2.3.3.3 Method Clear

```
[id(3)] HRESULT Clear();
```

2.3.3.4 The method ***Clear*** shall clean the message buffer.

2.4 Interface "TestResult"

2.4.1 Description

2.4.1.1 The identification of the interface shall be "DatabaseServer.TestResult".

2.4.2 Design

2.4.2.1 This interface shall be a dispatch interface.

2.4.2.2 This interface shall be an automation interface.

2.4.2.3 This interface shall implement the collection interface for *LPDISPATCH*, returning the *TestInfo* interfaces in the collection.

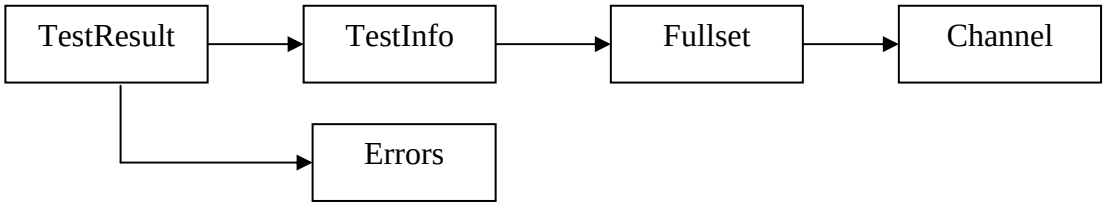

Figure 1: "*TestResult*" interface hierarchy

2.4.3 Methods and Properties

2.4.3.1 Property Filter

```
// Get and Set a filter used to retrieve only the
// Test information associated to the filtered parameters
[propget, id(4)
HRESULT Filter([in] BSTR FilterName, [out, retval] BSTR *pVal);
[propput, id(4)]
HRESULT Filter([in] BSTR FilterName, [in] BSTR newVal);
```

Argument Name	Description
FilterName	the name of the Filter Parameter. The valid values are “TestCellName”, “TestName”, “EngineType”, “EngineStandard”, “Build”, “SerialNumber”, “Customer” and “TestId”.
*pVal, newVal	a string representing one of the TestInfo parameters in the TRSCDB. A value of “” shall indicate that no filter is set for the given parameter.

2.4.3.1.1 It shall identify the filters used to narrow the test information search to the given parameters.

2.4.3.1.2 The property ***Filter*** string shall be case sensitive.

2.4.3.1.3 The property ***Filter*** strings shall be additive.

2.4.3.2 Property DatabaseName

```
// Return the name of the currently configured TRSCDB
[propget, id(2)]
HRESULT DatabaseName([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the name of the currently configured TRSCDB in the ODBC driver.

2.4.3.3 Method ResetAllFilters

```
// Note: all filters are additive (i.e. logical AND)
// Reset all filter.
[id(3)] HRESULT ResetAllFilters();
```

2.4.3.3.1 This method resets all filters so that all Test Information can be retrieved from the TRSCDB.

2.4.3.4 Method RetrieveTestInfo

```
// Retrieve all valid Test Information from the database
      // using the current filter,
[id(5)] HRESULT RetrieveTestInfo();
```

2.4.3.4.1 This method retrieves all valid Test Information from the TRSCDB, i.e. it shall request Test Information from the database. The Test Information shall be kept in the memory of the server. It will be possible to access the Test Information using the *TestInfo* property.

2.4.3.5 Method Identify

```
// Request the access to the Test Result Database in write mode.
[id(6)] HRESULT Identify (BSTR sUserName, BSTR sPassword);
```

Argument Name	Description
sUserName	The name of the user.
sPassword	The password associated to the user name

2.4.3.5.1 This method shall then retrieve the user level from the user security system (cf. 1.3). If this is not successful the user level shall be set to zero, i.e. only read access shall be possible.

2.4.3.6 Property Errors

```
// Get a pointer to the Errors interface.
[propget, id(7)]
HRESULT Errors([out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
*pVal	A Pointer to the Errors interface.

2.4.4 Usage Conditions and Restrictions

2.4.4.1 All Filters are reset at initialisation time.

2.4.4.2 The Filter properties will have to be set before taking effect in the *RetrieveTestInfo* method or in the collection.

2.4.4.3 The *RetrieveTestInfo* method will have to be called before accessing the *collection* properties for *TestInfo*.

2.5 Interface "TestInfo"

2.5.1 Description

2.5.1.1 This interface shall not be accessible directly. It shall be accessed through the *TestResult* interface using the *get_Item* property.

2.5.2 Design

2.5.2.1 This interface shall be a dispatch interface.

2.5.2.2 This interface shall be an automation interface.

2.5.2.3 This interface shall implement the collection interface for *LPDISPATCH*, returning the *Fullset* interfaces in the collection.

2.5.3 Methods and Properties

2.5.3.1 Property TestCellName

```
// Get the test name associated to the current Test.
[propget, id(2)]
HRESULT TestCellName([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the test cell name associated to the current test.

2.5.3.2 Property TestId

```
// Get the test identifier associated to the current Test.
[propget, id(3)]
HRESULT TestId([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the test identifier associated to the current Test.

2.5.3.3 Property TestName

```
// Get the test name associated to the current Test.
[propget, id(4)]
HRESULT TestName([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the test name associated to the current test.

2.5.3.4 Property TestDate

```
// Get the test date associated to the current Test.
[propget, id(5)]
HRESULT TestDate([out, retval] DATE *pVal);
```

Argument Name	Description
*pVal	the date and time of the beginning of the test.

2.5.3.5 Property TestDescription

```
// Get the test description associated to the current Test.
[propget, id(6)]
HRESULT TestDescription([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the test description associated to the current test.

2.5.3.6 Property EngineType

```
// Get the engine type associated to the current Test.
[propget, id(7)]
HRESULT EngineType([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the engine type associated to the current test.

2.5.3.7 Property EngineStandard

```
// Get the engine standard associated to the current Test.
[propget, id(8)]
HRESULT EngineStandard([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the engine standard associated to the current test.

2.5.3.8 Property Build

```
// Get the engine build associated to the current Test.
[propget, id(9)]
HRESULT Build([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the engine build number associated to the current test.

2.5.3.9 Property SerialNumber

```
// Get the engine serial number associated to the current Test.
[propget, id(10)]
HRESULT SerialNumber([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the engine serial number associated to the current test.

2.5.3.10 Property Customer

```
// Get the customer name associated to the current Test.
[propget, id(11)]
HRESULT Customer([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the customer name associated to the current test.

2.5.3.11 Property TestOperator1

```
// Get the name of the first test operator associated to
// the current Test.
[propget, id(12)]
HRESULT TestOperator1([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the name of the first test operator associated to the current test.

2.5.3.12 Property TestOperator2

```
// Get the name of the second test operator associated to
// the current Test.
[propget, id(13)]
HRESULT TestOperator2([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the name of the second test operator associated to the current test.

2.5.3.13 Property TestEngineer

```
// Get the name of the test engineer associated to the
// current Test.
[propget, id(14)]
HRESULT TestEngineer([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the name of the test engineer associated to the current test.

2.5.3.14 Property IsDataAvailable

```
// Return a boolean that indicates if the test data is
      // available in the database or not.
[propget, id(15)]
HRESULT IsDataAvailable([out, retval] BOOL *pVal);
```

Argument Name	Description
*pVal	True if the current test data is in the database, otherwise False.

2.5.3.14.1 Note that archive information is omitted since this interface will not be used to retrieve archived data from archive media.

2.5.3.15 Method RetrieveFullsets

```
// Retrieve all Fullset data associated to the current
// test information from the database.
[id(16)] HRESULT RetrieveFullsets();
```

2.5.3.15.1 This method shall retrieve all Fullset Information associated to the current Test Information from the TRSCDB, i.e. it shall request all Events of type Fullset and all channels associated to each event from the database. The Fullset Information shall be kept in the memory of the server. It will be possible to access the Fullset Information using the *get_Item* property.

2.5.3.16 Property AddFullset

```
// Add a new Fullset into the database
 [propget, id(17)] HRESULT AddFullset([in] LPDISPATCH newVal,
[in] long newConfigId, [out, retval] long *pEventId);
```

Argument Name	Description
newVal	A new Fullset to be stored in the TRSCDB. This is a Fullset sub-interface.
`newConfigId`	The Config ID assigned to the new fullset
`*pEventId`	The Event ID of the new fullset

2.5.3.17 Method UpdateFullset

```
// Update a Fullset into the database
[id(18)] HRESULT UpdateFullset(LPDISPATCH newVal);
```

Argument Name	Description
newVal	an updated Fullset to be stored in the TRSCDB. It is a Fullset sub-interface.

2.5.3.18 Property CustomerSpecific

```
// Get the Customer Specific information.
[propget, id(19)]
HRESULT CustomerSpecific([in} BSTR sPropertyName,
                         [out, retval] BSTR *pVal);
```

Argument Name	Description
sPropertyName	identify a unique customer specific property in the TRSCDB.
*pVal	the customer specific value associated to the *sPropertyName*.

2.5.3.19 Property TestCellId

```
// Get the test cell identifier associated to the current Test
// Cell.
[propget, id(20)]
HRESULT TestCellId([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the test cell identifier associated to the current Test Cell.

2.5.4 Usage Conditions and Restrictions

2.5.4.1 All properties shall be local, i.e. the properties apply only to one client.

2.5.4.2 The *RetrieveFullset* method will have to be called before accessing the collection of Fullsets.

2.5.4.3 The *get_Item* property will have to be called before calling the *AddFullset* method.

2.5.4.4 The *get_Item* property will have to be called before calling the *UpdateFullset* method.

2.6 Interface "Fullset"

2.6.1 Description

2.6.1.1 This interface shall not be accessible directly. It shall be accessed through the *TestInfo* interface using the *get_Item* property.

2.6.2 Design

2.6.2.1 This interface shall be a dispatch interface.

2.6.2.2 This interface shall be an automation interface.

2.6.2.3 This interface shall implement the collection interface for *LPDISPATCH*, returning the *Channel* interfaces in the collection.

2.6.3 Methods and Properties

2.6.3.1 Property ConfigId

```
// Get the Configuration Identifier.
[propget, id(2)]
HRESULT ConfigId([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the Configuration Identifier associated to a Fullset.

2.6.3.2 Property EventComment

```
// Get and Set the comment for an event.
[propget, id(3)]
HRESULT EventComment([out, retval] BSTR *pVal);
[propput, id(3)]
HRESULT EventComment([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	the Event Comment associated to a Fullset.

2.6.3.3 Property EventDate

```
// Get the date for an event.
[propget, id(4)]
HRESULT EventDate([out, retval] DATE *pVal);
```

Argument Name	Description
*pVal	the date when the fullset has been taken.

2.6.3.4 Property TestStep

```
// Get the Test Step description.
[propget, id(5)]
HRESULT TestStep([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the Test Step associated to a Fullset.

2.6.3.5 Property FullsetDuration

```
// Get the duration of a Fullset
[propget, id(6)]
HRESULT FullsetDuration([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the duration of a Fullset in seconds.

2.6.3.6 Property EngineeringUnits

```
// Get and Set the unit names
[propget, id(7)]
HRESULT EngineeringUnits([out, retval] VARIANT *pArrayVal);
[propput, id(7)]
HRESULT EngineeringUnits([in] VARIANT newArrayVal);
```

Argument Name	Description
*pArrayVal, newArrayVal	an array of engineering unit names of size *Count* associated to each channel in a Fullset.

2.6.3.7 Property Qualities

```
// Get and Set the quality flags
[propget, id(8)]
HRESULT Qualities([out, retval] VARIANT *pArrayVal);
[propput, id(8)]
HRESULT Qualities([in] VARIANT newArrayVal);
```

Argument Name	Description
*pArrayVal, newArrayVal	an array of quality status of size *Count* associated to each channel in a Fullset. The following values shall be valid: ”GOOD”, ”SUSPECT” and ”BAD”.

2.6.3.8 Property Names

```
// Get the channel names
[propget, id(9)]
HRESULT Names([out, retval] VARIANT *pArrayVal);
```

Argument Name	Description
*pArrayVal	an array of channel names of size *Count* associated to each channel in a Fullset.

2.6.3.9 Property Values

```
// Get and Set the Fullset Values
[propget, id(10)]
HRESULT Values([out, retval] VARIANT *pArrayVal);
[propput, id(10)]
HRESULT Values([in] VARIANT newArrayVal);
```

Argument Name	Description
*pArrayVal, newArrayVal	an array of channel values of size *Count* associated to each channel in a Fullset.

2.6.3.10 Property Find

```
[propget, id(11)]
HRESULT Find([in] BSTR sChannelName,
             [out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
sChannelName	A channel name in the fullset.
*pVal	If found, a channel interface, otherwise NULL.

2.6.3.11 Property EventId

```
// Get the Event Identifier
[propget, id(12)]
HRESULT EventId ([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the Event Identifier.

2.6.3.12 Property FullsetIndex

```
// Get the Fullset Index
[propget, id(13)]
HRESULT FullsetIndex ([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the Fullset Index, i.e. the value of FS_INDEX in the database table EVENTS.

2.6.4 Usage Conditions and Restrictions

2.6.4.1 All properties shall be local, i.e. the properties apply only to one client.

2.6.4.2 The number of defined names in *Names*, values in *Values*, qualities in *Qualities* and unit names in *EngineeringUnits* will have to be identical. The behaviour of the system will be unpredictable if the number of elements in the arrays differs.

2.6.5 Example

2.6.5.1 The following is an example of a VBS client.

```
' Establish the connection to the database server.
set test = CreateObject ("DatabaseServer.TestResult")

on error resume next

sUserName = ""
sPassword = ""
sScriptName = "UpdateFullset"

msgText = "Welcome to Update Fullset script!"  & vbLf & vbLf & _
        "You're connected to the Database: " & _
        test.DatabaseName & vbLf
MyMsgBox msgText
msgText = ""


sUserName = InputBox("Enter the User Name: ", sScriptName, "")
if (sUserName = vbEmpty) then
    set test = nothing
    WScript.quit
end if

sPassword = InputBox("Enter the User Password: ", sScriptName, "")
if (sPassword = vbEmpty) then
    set test = nothing
    WScript.quit
end if

test.Identify sUserName, sPassword
VerifyError "Identify"

sTestCellName = ""
sTestName = ""
sFullsetComment = ""
sChannelName = ""
sChannelValue = ""

sTestCellName = InputBox("Enter a Test Cell Name: ",_
                              sScriptName, "")
if (sTestCellName = vbEmpty) then
    set test = nothing
    WScript.quit
end if

sTestName = InputBox("Enter a Test Name: ",_
                         sScriptName, "")
if (sTestName = vbEmpty) then
    set test = nothing
    WScript.quit
```

```
end if

sFullsetComment = InputBox(_
      "Enter a Fullset Comment to be updated: ",_
      sScriptName, "")
if (sFullsetComment = vbEmpty) then
    set test = nothing
    WScript.quit
end if

sChannelName = InputBox("Enter a Channel Name: ",_
                              sScriptName, "")
if (sChannelName = vbEmpty) then
    set test = nothing
    WScript.quit
end if

sChannelValue = InputBox("Enter the new Channel's value: ",_
                               sScriptName, "")
if (sChannelValue = vbEmpty) then
    set test = nothing
    WScript.quit
end if

test.ResetAllFilters
VerifyError "ResetAllFilters"

test.Filter( "TestCellName" ) = sTestCellName
VerifyError "Filter"

test.Filter( "TestName" ) = sTestName
VerifyError "Filter"

' Retrieve the Test Information
test.RetrieveTestInfo
VerifyError "RetrieveTestInfo"

if (test.Count > 0) then
    if (test.Count > 1) then
      msgText = "Retrieved Test Information." & vbLf & vbLf
      msgText = msgText & "Found " & test.Count & _
                    " Test matching the current filter." & vbLf & vbLf
      MyMsgBox msgText
      msgText = ""

      testInfoCount = 0

      ' Go through the Test Information Array.
      for each testInfo in test
            testInfoCount = testInfoCount + 1

            ' display the current test information
            DisplayTestInfo testInfoCount, testInfo
      Next
```

```
      if (testInfoCount < test.Count) then
            VerifyError "Test(x)"
      end if
   else
      set testInfo = test(1)
      VerifyError "Test(1)"

      ' display the test information
      DisplayTestInfo 1, testInfo
   end if
else
   msgText = "Sorry!         " & vbLf & vbLf & _
             "The combination of           " & vbLf & _
             "     Test cell name: " & sTestCellName & vbLf & _
             "     Test name: " & sTestName & vbLf & vbLf & _
             "couldn't be found.                "
   MyMsgBox msgText
   msgText = ""
end if

set test = nothing
MyMsgBox "Done!"

'---------------------------------------------------------------------
' Verify if an error occured on the COM interface
Sub VerifyError (functionCalled)
   if (err.Number <> 0) then
      set errorList = test.Errors

      if (errorList.IsEmpty) then
            msgText = "ERROR!       " & vbLf & vbLf & _
                              "in " & functionCalled
            MyMsgBox msgText
            msgText = ""
      else
            do
                  MyMsgBox errorList.Message
            loop until errorList.IsEmpty
      end if

      WScript.quit
   end if
end sub

'---------------------------------------------------------------------
' Display a message box to user with option to cancel the script.
Sub MyMsgBox (sMessage)
   result = MsgBox(sMessage, vbOk, sScriptName)
   if (result = vbCancel) then
      WScript.quit
   end if
end sub

'---------------------------------------------------------------------
```

```
' Display the current test info information
Sub DisplayTestInfo (testInfoNum, testInfo)

    on error resume next

    if (testInfo.IsDataAvailable = 0) then
       msgText = "Test information # " & testInfoNum & vbLf & vbLf
       msgText = msgText & "No Data available for this test!" & vbLf
       msgText = msgText & "The Test Data has been archived." & vbLf

       MyMsgBox msgText
    else
       VerifyError "IsDataAvailable"

       msgText = "Test information # " & testInfoNum & vbLf & vbLf
       msgText = msgText & "Test Cell Name: " & testInfo.TestCellName _
                    & vbLf
       VerifyError "TestCellName"
       msgText = msgText & "Test Name: " & testInfo.TestName & vbLf
       VerifyError "TestName"
       msgText = msgText & "Test Date: " & testInfo.TestDate & vbLf _
                    & vbLf
       VerifyError "TestDate"
       msgText = msgText & "Test Description: " _
                    & testInfo.TestDescription & vbLf
       VerifyError "TestDescription"
       msgText = msgText & "Engine Type: " & testInfo.EngineType & vbLf
       VerifyError "EngineType"
       msgText = msgText & "Engine Standard: " _
                    & testInfo.EngineStandard & vbLf
       VerifyError "EngineStandard"
       msgText = msgText & "Build: " & testInfo.Build & vbLf
       VerifyError "Build"
       msgText = msgText & "Serial Number: " & testInfo.SerialNumber _
                    & vbLf
       VerifyError "SerialNumber"
       msgText = msgText & "Customer: " & testInfo.Customer & vbLf
       VerifyError "Customer"
       msgText = msgText & "Test Operator 1: " _
                    & testInfo.TestOperator1 & vbLf
       VerifyError "TestOperator1"
       msgText = msgText & "Test Operator 2: " _
                    & testInfo.TestOperator2 & vbLf
       VerifyError "TestOperator2"
       msgText = msgText & "Test Engineer: " _
                    & testInfo.TestEngineer & vbLf & vbLf
       VerifyError "TestEngineer"

       testInfo.RetrieveFullsets
       VerifyError "RetrieveFullsets"

       msgText = msgText & "Found " & testInfo.Count _
                    & " Fullset(s) for the current test information." _
                    & vbLf & vbLf
```

```
      VerifyError "Fullset Count"

      MyMsgBox msgText

      fullsetCount = 0

      ' Go through the Fullset Array.
      for each fullset in testInfo
            if (fullset.EventComment = sFullsetComment) then
                  fullsetCount = fullsetCount + 1

                  UpdateFullset testInfo, fullsetCount, fullset
                  VerifyError "UpdateFullset"
            end if
      next

      if (testInfo.Count > 0) then
            if (fullsetCount = 0) then
                  msgText = msgText & "Fullset comment: '" & sFullsetComment _
                                          & "' Not Found." & vbLf & vbLf
            end if
      end if

      testInfo.RetrieveFullsets
      VerifyError "RetrieveFullsets"

      msgText = msgText & "New Fullset Count " & testInfo.Count _
                    & vbLf & vbLf
      VerifyError "Fullset Count"

      ' Go through the Fullset Array.
      for each fullset in testInfo
            fullsetCount = fullsetCount + 1

            ' display the fullset information
            DisplayFullset fullsetCount, fullset
      next

      MyMsgBox msgText
   end if
end sub

'---------------------------------------------------------------------
' Display the fullset information
Sub DisplayFullset(fullsetNum, currFullset)

   on error resume next

   msgText = msgText & currFullset.EventComment
   VerifyError "EventComment"
   msgText = msgText & ", " & currFullset.EventDate
   VerifyError "EventDate"
   msgText = msgText & ", " & currFullset.TestStep
   VerifyError "TestStep"
```

```
    msgText = msgText & ", " & currFullset.FullsetDuration
    VerifyError "FullsetDuration"
    msgText = msgText & ", #Channel = " & currFullset.Count & vbLf
    VerifyError "Channel Count"
    set channel = currFullset.Find( sChannelName )
    VerifyError "Find Channel"
    msgText = msgText & "     Name: " & channel.Name _
                      & " = " & channel.Value _
                      & " " & channel.EngineeringUnit _
                      & vbLf
    VerifyError "EngineeringUnit"
end sub

'---------------------------------------------------------------------
' Create a new fullset
Sub UpdateFullset(testInfo, fullsetNum, currFullset)

    on error resume next

    set channel = currFullset.Find( sChannelName )
    VerifyError "Find Channel"
    channel.Value = sChannelValue
    VerifyError "Set Value"

    testInfo.UpdateFullset(currFullset)
    VerifyError "Update Fullset"
end sub

'---------------------------------------------------------------------
```

2.7 Interface "Channel"

2.7.1 Description

2.7.1.1 This interface shall not be accessible directly. It shall be accessed through the *Fullset* interface using the *get_Item* or *get_Find* property.

2.7.2 Design

2.7.2.1 This interface shall be a dispatch interface.

2.7.2.2 This interface shall be an automation interface.

2.7.3 Methods and Properties

2.7.3.1 Property Name

```
// Get the channel name.
[propget, id(1)]
HRESULT Name([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the channel name associated to a Channel.

2.7.3.2 Property Value

```
// Get and Set the value of the channel.
[propget, id(2)]
HRESULT Value([out, retval] double *pVal);
[propput, id(2)]
HRESULT Value([in] double newVal);
```

Argument Name	Description
*pVal, newVal	the channel value associated to a Channel.

2.7.3.3 Property EngineeringUnit

```
// Get and Set the engineering Unit Name of the channel.
[propget, id(3)]
HRESULT EngineeringUnit([out, retval] BSTR *pVal);
[propput, id(3)]
HRESULT EngineeringUnit ([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	the channel engineering unit name associated to a Channel.

2.7.3.4 Property Quality

```
// Get and Set the quality of the channel.
[propget, id(4)]
HRESULT Quality([out, retval] BSTR *pVal);
[propput, id(4)]
HRESULT Quality([in] BSTR newVal);
```

Argument Name	Description
*pVal, newVal	the channel quality associated to a Channel. The following values shall be a valid: “GOOD”, “SUSPECT” and “BAD”.

2.7.4 Usage Conditions and Restrictions

2.7.4.1 All properties shall be local, i.e. the properties apply only to one client.

2.7.5 Example

2.7.5.1 The following is an example of a VBS client.

```
' Establish the connection to the database server.
set test = CreateObject ("DatabaseServer.TestResult")

on error resume next

sTestCellName = ""
sTestName = ""
sChannelName = ""
sScriptName = "ListChannels"

msgText = "Welcome to List Channels script!"  & vbLf & vbLf & _
        "You're connected to the Database: " & _
        test.DatabaseName & vbLf
MyMsgBox msgText
```

```
sTestCellName = InputBox("Enter a Test Cell Name: ", sScriptName, "")
if (sTestCellName = vbEmpty) then
    set test = nothing
    WScript.quit
end if

sTestName = InputBox("Enter a Test Name: ", sScriptName, "")
if (sTestName = vbEmpty) then
    set test = nothing
    WScript.quit
end if

sChannelName = InputBox("Enter a Channel Name: ", sScriptName, "")
if (sChannelName = vbEmpty) then
    set test = nothing
    WScript.quit
end if

msgText = ""

test.ResetAllFilters
VerifyError "ResetAllFilters"

test.Filter( "TestCellName" ) = sTestCellName
VerifyError "Filter"

test.Filter( "TestName" ) = sTestName
VerifyError "Filter"

' Retrieve the Test Information
test.RetrieveTestInfo
VerifyError "RetrieveTestInfo"

if (test.Count > 0) then
    if (test.Count > 1) then
      msgText = "Retrieved Test Information." & vbLf & vbLf
      msgText = msgText & "Found " & test.Count & _
                    " Test matching the current filter." & vbLf & vbLf
      MyMsgBox msgText
      msgText = ""

      testInfoCount = 0

      ' Go through the Test Information Array.
      for each testInfo in test
            testInfoCount = testInfoCount + 1

            ' display the current test information
            DisplayTestInfo testInfoCount, testInfo
      Next
      if (testInfoCount < test.Count) then
            VerifyError "Test(x)"
      end if
    else
```

```
      set testInfo = test(1)
      VerifyError "Test(1)"

      ' display the test information
      DisplayTestInfo 1, testInfo
   end if
else
   msgText = "Sorry!         " & vbLf & vbLf & _
                "The combination of           " & vbLf & _
                "     Test cell name: " & sTestCellName & vbLf & _
                "     Test name: " & sTestName & vbLf & vbLf & _
                "couldn't be found.                "
   MyMsgBox msgText
   msgText = ""
end if

set test = nothing
MyMsgBox "Done!"

'---------------------------------------------------------------------
' Verify if an error occured on the COM interface
Sub VerifyError (functionCalled)
   if (err.Number <> 0) then
      set errorList = test.Errors

      if (errorList.IsEmpty) then
            msgText = "ERROR!       " & vbLf & vbLf & _
                              "in " & functionCalled
            MyMsgBox msgText
            msgText = ""
      else
            do
                  MyMsgBox errorList.Message
            loop until errorList.IsEmpty
      end if

      WScript.quit
   end if
end sub

'---------------------------------------------------------------------
' Display a message box to user with option to cancel the script.
Sub MyMsgBox (sMessage)
   result = MsgBox(sMessage, vbOk, sScriptName)
   if (result = vbCancel) then
      set test = nothing
      WScript.quit
   end if
end sub

'---------------------------------------------------------------------
' Display the current test info information
Sub DisplayTestInfo (testInfoNum, testInfo)
```

```
on error resume next

if (testInfo.IsDataAvailable = 0) then
   msgText = "Test information # " & testInfoNum & vbLf & vbLf
   msgText = msgText & "No Data available for this test!" & vbLf
   msgText = msgText & "The Test Data has been archived." & vbLf

   MyMsgBox msgText
else
   VerifyError "IsDataAvailable"

   msgText = "Test information # " & testInfoNum & vbLf & vbLf
   msgText = msgText & "Test Cell Name: " & testInfo.TestCellName _
                & vbLf
   VerifyError "TestCellName"
   msgText = msgText & "Test Name: " & testInfo.TestName & vbLf
   VerifyError "TestName"
   msgText = msgText & "Test Date: " & testInfo.TestDate & vbLf _
                & vbLf
   VerifyError "TestDate"
   msgText = msgText & "Test Description: " _
                & testInfo.TestDescription & vbLf
   VerifyError "TestDescription"
   msgText = msgText & "Engine Type: " & testInfo.EngineType & vbLf
   VerifyError "EngineType"
   msgText = msgText & "Engine Standard: " _
                & testInfo.EngineStandard & vbLf
   VerifyError "EngineStandard"
   msgText = msgText & "Build: " & testInfo.Build & vbLf
   VerifyError "Build"
   msgText = msgText & "Serial Number: " & testInfo.SerialNumber _
                & vbLf
   VerifyError "SerialNumber"
   msgText = msgText & "Customer: " & testInfo.Customer & vbLf
   VerifyError "Customer"
   msgText = msgText & "Test Operator 1: " _
                & testInfo.TestOperator1 & vbLf
   VerifyError "TestOperator1"
   msgText = msgText & "Test Operator 2: " _
                & testInfo.TestOperator2 & vbLf
   VerifyError "TestOperator2"
   msgText = msgText & "Test Engineer: " _
                & testInfo.TestEngineer & vbLf & vbLf
   VerifyError "TestEngineer"

   testInfo.RetrieveFullsets
   VerifyError "RetrieveFullsets"

   msgText = msgText & "Found " & testInfo.Count _
                & " Fullset(s) for the current test information." _
                & vbLf & vbLf
   VerifyError "Fullset Count"

   fullsetCount = 0
```

```
      ' Go through the Fullset Array.
      for each fullset in testInfo
            fullsetCount = fullsetCount + 1

            ' display the fullset information
            DisplayFullset fullsetCount, fullset
      next

      MyMsgBox msgText
   end if
end sub

'---------------------------------------------------------------------
' Display the fullset information
Sub DisplayFullset(fullsetNum, currFullset)

   on error resume next

   msgText = msgText & currFullset.EventComment
   VerifyError "EventComment"
   msgText = msgText & ", " & currFullset.EventDate
   VerifyError "EventDate"
   msgText = msgText & ", " & currFullset.TestStep
   VerifyError "TestStep"
   msgText = msgText & ", " & currFullset.FullsetDuration
   VerifyError "FullsetDuration"
   msgText = msgText & ", #Channel = " & currFullset.Count & vbLf
   VerifyError "Channel Count"
   set channel = currFullset.Find( sChannelName )
   VerifyError "Find Channel"
   msgText = msgText & "     Name: " & channel.Name _
                     & " = " & channel.Value _
                     & " " & channel.EngineeringUnit _
                     & vbLf
   VerifyError "EngineeringUnit"
end sub

'---------------------------------------------------------------------
```

2.8 Interface "SensorCal"

2.8.1 Description

2.8.1.1 The identification of the interface shall be "DatabaseServer.SensorCal".

2.8.2 Design

2.8.2.1 This interface shall be a dispatch interface.

2.8.2.2 This interface shall be an automation interface.

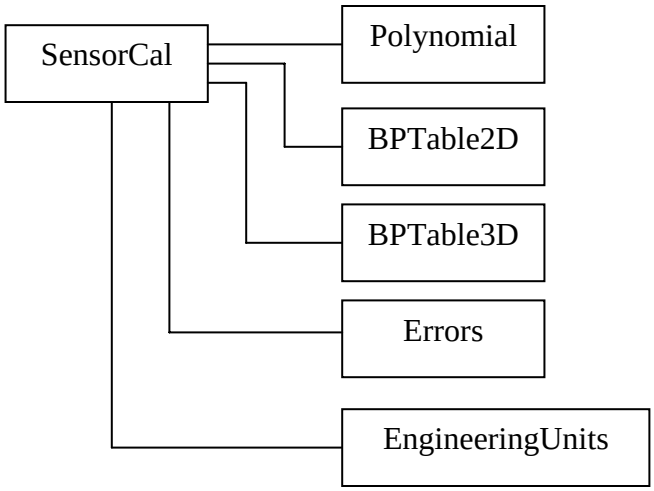

Figure 2: "*SensorCal*" interface hierarchy

2.8.3 Methods and Properties

2.8.3.1 Property Select

```
// Find the sensor name.
[propget, id(1)] HRESULT Select(BSTR sSensorName, BOOL *pVal);
```

Argument Name	Description
sSensorName	The name of the sensor to be retrieved from the TRSCDB.
*pVal	True if the sensor name could be found, otherwise False.

2.8.3.1.1 The sensor information shall be kept in memory. It will be possible to access the sensor information using the properties defined below.

2.8.3.2 Property SensorName

```
// Get the Sensor Name
[propget, id(2)]
HRESULT SensorName([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the name of the selected sensor dot separated.

2.8.3.3 Property EngineeringUnitOut

```
// Get the Output Unit Identifier
[propget, id(4)]
HRESULT EngineeringUnitOut([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the Output Unit Name for the selected sensor.

2.8.3.4 Property EngineeringUnitIn

```
// Get the Input Unit Identifier
[propget, id(5)]
HRESULT EngineeringUnitIn([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the Input Unit Name for the selected sensor.

2.8.3.5 Property RangeMin

```
// Get the minimum range value for the calibration data
[propget, id(6)]
HRESULT RangeMin([out, retval] double *pVal);
```

Argument Name	Description
*pVal	the minimum allowed value for the selected sensor.

2.8.3.6 Property RangeMax

```
// Get the maximum range value for the calibration data
[propget, id(7)]
HRESULT RangeMax([out, retval] double *pVal);
```

Argument Name	Description
*pVal	the maximum allowed value for the selected sensor.

2.8.3.7 Property CalibrationDate

```
// Get the date when the calibration was performed
[propget, id(8)]
HRESULT CalibrationDate([out, retval] DATE *pVal);
```

Argument Name	Description
*pVal	the date when the selected sensor was calibrated.

2.8.3.8 Property DueDate

```
// Get the date when the calibration is due
[propget, id(9)]
HRESULT DueDate([out, retval] DATE *pVal);
```

Argument Name	Description
*pVal	the date when the selected sensor is due for the next calibration.

2.8.3.9 Property TypeId

```
// Get the calibration data Type Identifier
[propget, id(10)]
HRESULT TypeId([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the type of calibration information for the selected sensor: “POLY” for Polynomial, “2D” for 2D Break Point Table, ”3D” for 3D Break Point Table.

2.8.3.10 Property Data

```
// Get descriptor to the calibration data interface
// Polynomial, BPTable2D or BPTable3D
[propget, id(11)]
HRESULT Data([out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
*pVal	a pointer to the sub-interface "*Polynomial*", "*BPTable2D*" or "*BPTable3D*".

2.8.3.10.1 The returned sub-interface depends on the *TypeId* of the sensor calibration. The property ***Data*** will return a "*Polynomial*" sub-interface if the *TypeId* is set to "POLY" or a "*BPTable2D*" sub-interface if the *TypeId* is set to "2D" or a "*BPTable3D*" sub-interface if the *TypeId* is set to "3D".

2.8.3.11 Property RawTypeId

```
// Get the Raw data Type Identifier
[propget, id(12)]
HRESULT RawTypeId([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the type of raw data information for the selected sensor: "POLY" for Polynomial, "2D" for 2D Break Point Table, "3D" for 3D Break Point Table. If the *RawTypeId* is not defined an empty string will be returned.

2.8.3.12 Property RawData

```
// Get descriptor to the raw calibration data interface
// Polynomial, BPTable2D or BPTable3D
[propget, id(13)]
HRESULT RawData([out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
*pVal	a pointer to the sub-interface "*Polynomiall*", "*BPTable2D*" or "*BPTable3D*". If no data is available a NULL pointer will be returned.

2.8.3.12.1 The returned sub-interface depends on the *RawTypeId* of the sensor calibration. The property ***RawData*** will return a "*Polynomial*" sub-interface if the *RawTypeId* is set to "POLY" or a "*BPTable2D*" sub-interface if the *RawTypeId* is set to "2D" or a "*BPTable3D*" sub-interface if the *RawTypeId* is set to "3D".

2.8.3.13 Property Errors

```
// Get a pointer to the Errors interface.
[propget, id(14)]
HRESULT Errors([out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
*pVal	A Pointer to the Errors interface.

2.8.3.14 Property EngineeringUnits

```
// Get a pointer to the EngineeringUnits interface.
[propget, id(15)]
HRESULT EngineeringUnits ([out, retval] LPDISPATCH *pVal);
```

Argument Name	Description
*pVal	A Pointer to the EngineeringUnits interface.

2.8.3.15 Property DatabaseName

```
// Return the name of the currently configured TRSCDB
[propget, id(16)]
HRESULT DatabaseName([out, retval] BSTR *pVal);
```

Argument Name	Description
*pVal	the name of the currently configured TRSCDB in the ODBC driver.

2.8.4 Usage Conditions and Restrictions

2.8.4.1 The *Select* property will have to be called before accessing the other properties (except for the *Errors*, *EngineeringUnits* and *DatabaseName* properties).

2.8.4.2 The properties ***TypeId*** and ***Data*** will be associated to the calibration data generated during the calibration of a Sensor.

2.8.4.3 The properties ***RawTypeId*** and ***RawData*** will be associated to the certificate data provided with a Sensor. This information may be empty.

2.8.5 Example

2.8.5.1 The following is an example of a VBS client.

```
set SensorCal = CreateObject ("DatabaseServer.SensorCal")

sSensorName = InputBox("Enter a sensor name:", _
                       "Sensor Name", "")

bFound = SensorCal.Select(sSensorName)

if (bFound = FALSE) then
    msgText = "The Sensor Name: " & sSensorName & _
                    " couldn't be found!" & vbLf & vbLf
    MyMsgBox msgText

    WScript.Quit
end if

msgText = "Sensor Information for: " & vbLf & vbLf
msgText = msgText & "Sensor Name    : " & SensorCal.SensorName & vbLf

msgText = msgText & "Calibration Date     : " _
                  & SensorCal.CalibrationDAte & vblf
msgText = msgText & "Due Date       : " & SensorCal.DueDAte & vblf
msgText = msgText & "Range Max      : " & SensorCal.RangeMax & vblf
msgText = msgText & "Range Min      : " & SensorCal.RangeMin & vblf
```

```
msgText = msgText & "Raw Type Id    : " & SensorCal.RawTypeId & vblf
msgText = msgText & "Type Id        : " & SensorCal.TypeId & vblf
msgText = msgText & "Unit Id In           : " & SensorCal.EngineeringUnitIn _
                                    & vblf
msgText = msgText & "Unit Id Out    : " & SensorCal. EngineeringUnitOut _
                                    & vblf & vbLf

' Display the calibration data (polynomial, 2D or 3D break point tables)
DisplayData SensorCal

MsgBox msgText

' Free the interface
set SensorCal = Nothing
```

2.9 Interface "Polynomial"

2.9.1 Description

2.9.1.1 This interface shall not be accessible directly. It shall be accessed through the *SensorCal* interface using the *Data* or *RawData* properties.

2.9.2 Design

2.9.2.1 This interface shall be a dispatch interface.

2.9.2.2 This interface shall be an automation interface.

2.9.3 Methods and Properties

2.9.3.1 Property Degree

```
// Get the polynomial degree
[propget, id(1)]
HRESULT Degree([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the size of the polynomial associated to the selected sensor, i.e. ***n*** in the polynomial equation $y = C_0 + C_1 x + C_2 x^2 + ... + C_n x^n$. It shall be initially set to 0.

2.9.3.2 Property Coefficient

```
// Get the coefficients Cn as in
//    y = C0 + C1 * x + C2 * x^2 + ... + Cn * x^n.
[propget, id(2)]
HRESULT Coefficient([in] long iOrder,
                    [out, retval] double *pVal);
```

Argument Name	Description
iOrder	the order of the coefficient to be retrieved
*pVal	the coefficient associated to the selected sensor.

2.9.4 Usage Conditions and Restrictions

2.9.4.1 All properties shall be local, i.e. the properties apply only to one client.

2.9.4.2 The parameter *iOrder* value of the *Coefficient* property will have to be between 0 and *Degree.*

2.9.5 Example

2.9.5.1 The following is an example of a VBS sub-procedure client:

```
'-----------------------------------------------------------------------
sub DisplayData SensorCal

  ' Polynomial
  if (SensorCal.TypeId = "POLY") then
    set poly = SensorCal.Data

    For iOrder = 0 to poly.Degree
      msgText = msgText & "Coef ” & iOrder & “       : " _
                        & poly.Coefficient(iOrder) & vbLf
    next
  end if
end sub

'-----------------------------------------------------------------------
```

2.10 Interface "BPTable2D"

2.10.1 Description

2.10.1.1 This interface shall not be accessible directly. It shall be accessed through the *SensorCal* interface using the *Data* or *RawData* properties.

2.10.2 Design

2.10.2.1 This interface shall be a dispatch interface.

2.10.2.2 This interface shall be an automation interface.

2.10.3 Methods and Properties

2.10.3.1 Property Size

```
// Get the size of the break point table
[propget, id(1)]
HRESULT Size([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the number of points defined in the *BPTable2D* interface. It shall be initially set to 0.

2.10.3.2 Property XCoord

```
// Get the coordinate X from the break point table.
[propget, id(2)]
      HRESULT XCoord([in] long iPos, [out, retval] double *pVal);
```

Argument Name	Description
iPos	identify the position (between 1 and *Size*) in the Break Point Table.
*pVal	the coordinate X associated to the selected sensor calibration information and position *iPos*.

2.10.3.3 Property YCoord

```
// Get the coordinate Y from the break point table.
[propget, id(3)]
HRESULT YCoord([in] long iPos, [out, retval] double *pVal);
```

Argument Name	Description
iPos	identify the position (between 1 and *Size*) in the Break Point Table.
*pVal	the coordinate Y associated to the selected sensor calibration information and position *iPos*.

2.10.4 Usage Conditions and Restrictions

2.10.4.1 All properties shall be local, i.e. the properties apply only to one client.

2.10.5 Example

2.10.5.1 The following is an example of a VBS sub-procedure client:

```
'--------------------------------------------------------------------------
sub DisplayData SensorCal

  ' 2D Break Point Table
  if (SensorCal.TypeId = "2D") then
    set BP2D = SensorCal.Data

    for sequenceNum = 1 to BP2D.Size
      msgText = msgText & "(X, Y)        : (" & BP2D.XCoord(sequenceNum)
      msgText = msgText & ", " & BP2D.YCoord(sequenceNum) & ")" & vbLf
    next
  end if
end sub
'--------------------------------------------------------------------------
```

2.11 Interface "BPTable3D"

2.11.1 Description

2.11.1.1 This interface shall not be accessible directly. It shall be accessed through the *SensorCal* interface using the *Data* or *RawData* properties.

2.11.2 Design

2.11.2.1 This interface shall be a dispatch interface.

2.11.2.2 This interface shall be an automation interface.

2.11.3 Methods and Properties

2.11.3.1 Property Size

```
// Get the size of the break point table
[propget, id(1)]
HRESULT Size([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the number of points defined in the *BPTable3D* interface. It shall be initially set to 0.

2.11.3.2 Property XCoord

```
// Get the coordinate X from the break point table.
[propget, id(2)]
      HRESULT XCoord([in] long iPos, [out, retval] double *pVal);
```

Argument Name	Description
iPos	identify the position (between 1 and *Size*) in the Break Point Table.
*pVal	the coordinate X associated to the selected sensor calibration information and position *iPos*.

2.11.3.3 Property YCoord

```
// Get the coordinate Y from the break point table.
[propget, id(3)]
HRESULT YCoord([in] long iPos, [out, retval] double *pVal);
```

Argument Name	Description
iPos	identify the position (between 1 and *Size*) in the Break Point Table.
*pVal	the coordinate Y associated to the selected sensor calibration information and position *iPos*.

2.11.3.4 Property ZCoord

```
// Get the coordinate Z from the break point table.
[propget, id(4)]
HRESULT ZCoord([in] long iPos, [out, retval] double *pVal);
```

Argument Name	Description
iPos	identify the position (between 1 and *Size*) in the Break Point Table.
*pVal	the coordinate Z associated to the selected sensor calibration information and position *iPos*.

2.11.4 Usage Conditions and Restrictions

2.11.4.1 All properties shall be local, i.e. the properties apply only to one client.

2.11.5 Example

2.11.5.1 The following is an example of a VBS sub-procedure client:

```
'---------------------------------------------------------------------
sub DisplayData(SensorCal)

  ' 3D Break Point Table
  if (SensorCal.TypeId = "3D") then
    set BP3D = SensorCal.Data

    for iPos = 1 to BP3D.Size
      msgText = msgText & "(X, Y, Z)     :"
      msgText = msgText & " (" & BP3D.XCoord(iPos)
      msgText = msgText & ", " & BP3D.YCoord(iPos)
      msgText = msgText & ", " & BP3D.ZCoord(iPos) & ")" & vbLf
    next
  end if
end sub

'---------------------------------------------------------------------
```

2.12 Interface "EngineeringUnits"

2.12.1 Description

2.12.1.1 This interface shall not be accessible directly. It shall be accessed through the *SensorCal* interface using the *EngineeringUnits* property.

2.12.2 Design

2.12.2.1 This interface shall be a dispatch interface.

2.12.2.2 This interface shall be an automation interface.

2.12.2.3 This interface shall implement a collection of strings, returning the *EngineeringUnit's name.*

2.12.3 Methods and Properties

2.12.3.1 Property Count

```
// Number of Engineering Units configured
[propget, id(1)]
HRESULT Count([out, retval] long *pVal);
```

Argument Name	Description
*pVal	the number of engineering units in the database.

2.12.3.2 Property Find

```
[propget, id(2)]
HRESULT Find([in] BSTR Name, [out, retval] long *pVal);
```

Argument Name	Description
Name	The unit's name to be found.
*pVal	If Found, return the position in the array of units, otherwise 0.

2.12.3.3 Method Add

```
[id(4)] HRESULT Add([in] BSTR Name);
```

Argument Name	Description
Name	Add the Engineering Unit *Name* in the database.

2.12.3.4 Method Clear

```
// Delete all Engineering Units.
[id(5)] HRESULT Clear();
```

2.12.4 Usage Conditions and Restrictions

2.12.4.1 All properties shall be local, i.e. the properties apply only to one client.