

## INTERFACE WITH SAS USING THIS .NET CLASS

Matthew Campbell, Educational Testing Service, Princeton, NJ

### Abstract

Situations often arise where having the ability to interface with SAS® in .NET® is extremely valuable. For instance, applications can be built with windows interfaces that utilize SAS code to generate reports. This paper will present a Visual Basic® class, a symbolic representation of a programming object, which has been used to work with the SAS system in many projects. This will include VB.NET code to open SAS, submit SAS code, and read data stored in permanent SAS datasets. In addition, an example of how this SAS class works with a .NET Windows® application will be presented.

### Introduction

Since version 6.12, SAS for Windows has been capable of operating as an OLE® automation server (Clegg & Rigsbee, 1997). This allows programming languages such as Visual Basic (VB) to be used as scripting tools to automate the SAS system. This paper demonstrates a VB class called SAS\_OLE that has been used to work with SAS as an automation server. The SAS\_OLE class was written with Visual Studio .NET 1.1 on Windows 2000 for SAS 8.2.

Each of the sections represents the major tasks the SAS\_OLE class can accomplish: opening up SAS, submitting code to SAS, running a SAS program, reading a SAS dataset, and using SAS from a windows application. Only the most important methods are discussed, to see the complete implementation of the class code see Appendix A or view the file SAS\_OLE.vb that is included with the conference proceedings.

### Opening Up SAS in Visual Basic .NET

Before the client application is able to use the SAS\_OLE class in code an object must be created. The syntax for creating a SAS\_OLE object in VB is this:

```
Dim SASDemo As New SAS_OLE(False)
```

This creates an object called SASDemo that will be used to invoke the methods and properties of the SAS\_OLE class throughout the client application. The False variable in the statement above is used to determine whether the SAS\_OLE class will be used just to access SAS datasets or if it will be used to utilize the entire SAS system.

In VB, the New method is a special procedure called a constructor. A constructor is called each time an object is created from a class. Since a constructor must be called to create an object, the New method is a good place to initialize objects that will be used throughout the class. In the SAS\_OLE class the New method creates a SAS OLE automation server (if DataOnly is set to False) and sets the values of the variables DataOnly and Disposed.

Here is the code for the New method:

```

Public Sub New(ByVal DataOnly As Boolean)
    Try
        Me.Disposed = False
        Me.m_DataOnly = DataOnly
        If Not Me.DataOnly Then
            Try
                Me.SAS_Server = CreateObject("SAS.Application")
                Me.SAS_Server.Wait = True
            Catch ex As Exception
                Dim SASError As New Exception("Error: SAS " & _
                    "is not installed on this system.", ex)
                Throw SASError
            End Try
        End If
    Catch ex As Exception
        Throw ex
    End Try
End Sub

```

The code here is enclosed in the “Try, Catch” syntax of VB. This allows for structured error handling. It works by attempting to execute code after the Try statement. If the code causes an error then it skips down to the Catch block where the error can be handled or thrown to the client application or procedure. This procedure has two Try blocks.

In the first Try block the code attempts to set two variables:

```

Try
    Me.Disposed = False
    Me.m_DataOnly = DataOnly

```

Disposed is a variable that indicates to the system whether the objects associated with the class have been released from memory. It is important for the memory management functions of VB. For more information about the Dispose method in VB, visit:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenre/html/cpconfinalizeddispose.asp>

DataOnly refers to whether the current session will be only used to read SAS datasets or if the entire SAS System needs to be used.

The next block of code attempts to create an instance of a SAS automation server if DataOnly was set to False:

```

If Not Me.DataOnly Then
    Try
        Me.SAS_Server = CreateObject("SAS.Application")
        Me.SAS_Server.Wait = True
    Catch ex As Exception
        Dim SASError As New Exception("Error: SAS " & _
            "is not installed on this system.", ex)
        Throw SASError
    End Try
End If

```

The lines of code that actually create the SAS session are:

```

Me.SAS_Server = CreateObject("SAS.Application")
Me.SAS_Server.Wait = True

```

This sets the SAS\_Server object, which is a SAS\_OLE class level object, to an instance of the SAS automation server. The second line of code sets the Wait property of SAS\_Server to True. The Wait property ensures that SAS will finish processing before checking for return values. Return values, such as RC and ResultString, allow information to be sent from within the SAS session to the client application. When the SAS\_Server object is created and the client application attempts to set the Wait property to True an error will be raised if SAS is not installed on the client computer.

```
Catch ex As Exception
    Dim SASError As New Exception("Error: SAS " & _
        "is not installed on this system.", ex)
    Throw SASError
```

Enclosing the code that created the SAS server object in the Try Catch block above allows the client program to catch the error when a user attempts to use this class without having SAS installed on their machine.

The properties and methods of the SAS session are accessed from the SAS\_OLE class code by typing the name SAS\_Server followed by a dot and the name of the property or method. For instance, to change the title that shows at the top of the SAS window the following code would be used:

```
SAS_Server.Title = "SAS Automation DEMO"
```

For more information on the properties and methods of the SAS automation server see *SAS Companion for the Microsoft Windows Environment, Version 8* (SAS Institute, 1999).

For the purposes of this paper, all calls to the SAS automation served are handled through the SAS\_OLE class.

### Submit SAS Code From Visual Basic

VB can utilize the SAS system to do tasks such as submitting code to SAS once an instance of the SAS\_OLE class has been created. The code `Dim SASDemo As New SAS_OLE(False)` discussed above created an object called SASDemo to represent the SAS\_OLE class in code. Once this has been done, the SASDemo object can use the methods of the SAS\_OLE class to submit code:

```
SASDemo.SubmitCode("proc print data = one;")
SASDemo.SubmitCode("run")
```

Here is the underlying SubmitCode method:

```
Public Overridable Sub SubmitCode(ByVal SASCode As String)
    Try
        If Me.DataOnly <> True Then
            SASCode = SASCode.Trim()
            If SASCode.Length > 196 Then
                Dim CodeError As New Exception("Code must be" & _
                    " less than 196 characters")
                Throw CodeError
            Exit Sub
        Else
            If SASCode <> "" Then
                Me.SAS_Server.Submit(SASCode)
            End If
        End Try
    End Sub
```

```

        End If
    End If
Else
    Dim CodeError As New Exception("SAS_OLE is not set" & _
        " up to work with SAS code. Run with DataOnly" & _
        " option set to True.")
    Throw CodeError
    Exit Sub
End If
Catch ex As Exception
    Throw ex
End Try
End Sub

```

The SubmitCode method allows the client application to handle two common errors associated with submitting SAS code in this way: attempting to submit code without a SAS session open and attempting to submit strings of more than 195 characters.

Through trial and error it was found that the submit method cannot handle more than 195 characters at once. As a result, the SubmitCode method of the SAS\_OLE class checks to make sure that its input string is no longer that 195 characters using the following code:

```

SASCode = SASCode.Trim()
If SASCode.Length > 196 Then
    Dim CodeError As New Exception("Code must be" & _
        " less than 196 characters")
    Throw CodeError
    Exit Sub

```

## Run a SAS Program From Visual Basic

Another useful task that the SAS\_OLE class can be used for is submitting complete SAS programs. This is done using the SubmitProgram method:

```
SASDemo.SubmitProgram("C:\Temp\SAS_Program.sas")
```

However, a problem arises when attempting to run more than one program at a time. If multiple %include statements are used to run SAS programs all will attempt to run at the same time. This can cause many problems, especially if some of the programs depend on others to finish processing.

To deal with this issue, SAS\_OLE includes a SubmitProgram method that allows the client application to submit timed programs:

```
SASDemo.SubmitProgram("C:\Temp\SAS_Program.sas", True)
```

The workaround that is used to alleviate this program takes advantage of the RC function. The RC function is a return code that can be set from within the SAS session and can also be read by the client program; the idea behind the workaround is that the RC function is set to 0 right before the program starts and then to 1 once the program finishes. The client application monitors the RC value and waits for it to flip to 1 before continuing to the next line of code.

The first step required to use the RC function in this way is to put code into the actual SAS program that is to be run:

```
data _null_;
    error=setrc("Start",0.0);
    put error;
run;
```

.....SAS Program Code.....

```
data _null_;
    error=setrc("End",1.0);
    put error;
run;
```

This code will set the RC function to 0 right before the program executes. Once the program code in the middle is done running the RC function will be set to 1.

The next step SubmitProgram needs to do is to submit the program, including code that will monitor the RC function:

```
Me.SubmitCode("%include '" & FileName & "';")
Do Until SAS_Server.rc = 1
Loop
```

.....lines of code.....

The first line submits the program with a %include statement. The next two lines are a loop which holds up the VB processing until the RC function is set back to 1. This will ensure that the program is done processing before going on to the next lines of code. One warning, however, is if the SAS program itself crashes (or if it never sets the RC function to 1) the client program will get stuck in an infinite loop.

Here is the entire method:

```
Public Sub SubmitProgram(ByVal FileName As String, _
                        ByVal WaitForProgramToFinish As Boolean)
Try
    FileName = AppPath & FileName
    FileName = FileName.Trim()
    If FileName <> "" Then
        If WaitForProgramToFinish Then
            Me.SubmitCode("data _null_;")
            Me.SubmitCode("error=setrc('Start',0.0);")
            Me.SubmitCode("put error;")
            Me.SubmitCode("run;")
            Me.SubmitCode("%include '" & FileName & "';")
            Do Until SAS_Server.rc = 1
            Loop
            Me.SubmitCode("data _null_;")
            Me.SubmitCode("error=setrc('Start',0.0);")
            Me.SubmitCode("put error;")
            Me.SubmitCode("run;")
        Else
            Me.SubmitCode("%include '" & FileName & "';")
        End If
    Else
        Dim CodeError As New Exception("SAS_OLE.Submit" & _
            "Program: Attempted to submit a SAS program with" & _
            "a blank filename")
        Throw CodeError
    Exit Sub
    End If
Catch ex As Exception
    Throw ex
End Try
End Sub
```

## Read SAS Datasets Into Visual Basic

Another task that is important for a client application is reading a SAS dataset directly into data tables that VB can work with. ADO.NET, the data access technology for .NET, makes it possible to convert a SAS dataset into a .NET data table. This is usually necessary once code has been executed and the results of the analysis need to be displayed or formatted into a document of some type.

The SAS\_OLE class implements this with the function DataSet. This is an example of how a client application would use the DataSet function:

```
Dim MyDataTable As DataTable
SASDemo.LibName = "C:\Temp"
MyDataTable = SASDemo.DataSet("one")
```

This populates the data table "MyDataTable" with data from the SAS dataset "one" in directory "C:\Temp". For more information on working with ADO datasets visit:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/html/vbconDataSets.asp>.

The steps that the SAS\_OLE class DataSet function uses to read a SAS dataset in .NET are:

- 1.) Create a connection to the directory where the datasets are located
- 2.) Create a command object with the connection above
- 3.) Create a data adapter with the command object above
- 4.) Create a dataset object
- 5.) Fill the dataset object using the data adapter

The first step is to create an ADO.NET connection to the directory where the SAS datasets are stored:

```
Dim Me.cnSASData As OleDbConnection

Me.cnSASData = New OleDbConnection("Provider=sas.LocalProvider.1;" & _
    "Data Source=C:\Temp")

Me.cnSASData.Open()
```

This created a connection object, set the data source to the temp directory, and finally opened the connection.

Next, create a command object using the connection from above and set the properties CommandType to TableDirect and CommandText to the name of the SAS dataset:

```
Dim cmd As OleDb.OleDbCommand = Me.cnSASData.CreateCommand
cmd.CommandType = CommandType.TableDirect
cmd.CommandText = "MY_DATASET"
```

Now, create a data adapter with the command object created above.

```
Dim da As OleDb.OleDbDataAdapter = New OleDb.OleDbDataAdapter(cmd)
```

Finally, declare a new dataset and fill it with the data adapter:

```
Dim ds As New DataSet
da.Fill(ds, "MY_DATASET")
```

Once the dataset is filled, the DataSet function will return the table that corresponds to the SAS dataset table. Here is how all the code above works to implement the DataSet function:

```
Public Overridable Function DataSet(ByVal DataSetName As String) _
    As DataTable

    Try
        'open data connection if it is closed:
        If Not Me.cnSASData Is Nothing Then
            If Me.cnSASData.State = ConnectionState.Closed Then
                Me.OpenLibName()
            End If
        Else
            Me.OpenLibName()
        End If

        'create command object:
        Dim cmd As OleDb.OleDbCommand = Me.cnSASData.CreateCommand
        cmd.CommandType = CommandType.TableDirect
        cmd.CommandText = DataSetName

        'create dataset, dataadapter and then fill with sas dataset
        'rows.
        Dim da As OleDb.OleDbDataAdapter = _
            New OleDb.OleDbDataAdapter(cmd)
        Dim ds As New DataSet
        Dim dt As New DataTable
        da.Fill(ds, DataSetName)

        'if successful return a copy of the datatable to the client
        'application:
        If ds Is Nothing Then
            Return Nothing
        Else
            If ds.Tables(0).Rows.Count = 0 Then
                Return Nothing
            Else
                dt = ds.Tables(0).Copy
                Return dt
            End If
        End If

    Catch ex As Exception
        Throw ex
    End Try
End Function
```

### Use the SAS\_OLE Class in a Visual Basic Windows Application

Now, using the SAS\_OLE class in a windows application will be demonstrated. This application will read the file names and file sizes from a windows directory, put the information into a SAS dataset, and display the results on a data grid on the windows form.

The first step is to create a new windows project in the Visual Studio IDE® and add a data grid and a command button from the IDE toolbox. Once this is done, the SAS\_OLE class needs to be added to the project. From the IDE main menu select "Project" and

then “Add Existing Item”. Browse to the location where you have the SAS\_OLE.vb file (which is included on the CD) and select the SAS\_OLE.vb file.

In the form load event, insert code that will create an instance of the SAS\_OLE class:

```
Dim SASDemo As New SAS_OLE(False)
```

This creates an object "SASDemo" that will be used to refer to the SAS\_OLE class. When the object was created a SAS automation session was also started. Next, use the SubmitCode method to set up a SAS libname and dataset in the SAS session:

```
SASDemo.SubmitCode ("libname Temp 'C:\Temp';")
SASDemo.SubmitCode ("data Temp.one;")
```

Use the VB DirectoryInfo class of the IO namespace to get information about the files in “C:\Temp” and the SASDemo object to insert the information into the dataset “one”.

```
For Each f As IO.FileInfo In New IO.DirectoryInfo("C:\Temp").GetFiles
    SASDemo.SubmitCode ("FilePath = '" & f.FullName & "';")
    SASDemo.SubmitCode ("FileSize = '" & f.Length & "';")
    SASDemo.SubmitCode ("output;")
Next
SASDemo.SubmitCode ("run;")
```

Finally, add the code below to the button's click event to use the SAS\_OLE DataSet function to display the dataset created above in a datagrid:

```
SASDemo.LibName = "C:\Temp"
Me.DataGrid1.DataSource = SASDemo.DataSet("one")
```

## Conclusion

Using VB and SAS together can provide good solutions to data analysis problems: adding statistical processing power to VB applications, automating SAS code through windows or web interfaces, and leveraging the strengths of VB and SAS when used together. Organizing the code that works with SAS in the form of a class such as the SAS\_OLE class demonstrated here is an appropriate solution to automating the SAS system.

Other solutions exist that allow the SAS system to be automated. Notably, SAS 9 includes the SAS IOM® (Integrated Object Model) that allows similar functionality to SAS\_OLE. This object library can be used in SAS version 8.2 as well, but it requires a special download. Some of the advantages of using the OLE automation are the SAS\_OLE class code can be used with versions of SAS starting at 6.12, no special downloads are required to be installed on user's computers, and no expensive licenses other than Base SAS are required for others who use applications built with OLE automation.

## REFERENCES

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration.

Windows®, Visual Basic® and .NET® and all other Microsoft product or service names are registered trademarks or trademarks of Microsoft in the USA and other countries. ® Indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

Clegg & Rigsbee, "OLE and the SAS System for Windows Release 6.12", Proceedings of the Twenty-Second Annual SAS Users Group international Conference, paper 46, 1997.

Microsoft, Inc., "Microsoft Solution Developers Network", <http://msdn.microsoft.com>, Redmond, WA, Microsoft 2003.

SAS Institute Inc., "Sample: SAS OLE DB Data Provider", <http://support.sas.com/ctx/samples/index.jsp?sid=1190>: SAS Institute Inc., 2003.

SAS Institute Inc., "SAS Companion for the Microsoft Windows Environment, Version 8", Cary, NC: SAS Institute Inc., 1999.

## CONTACT INFORMATION

Matthew Campbell  
Educational Testing Services  
Rosedale Road, NJ 08541  
Work Phone: (609) 683-2293  
Email: [mjcampbell@ets.org](mailto:mjcampbell@ets.org)

## APPENDIX A – SAS\_OLE CLASS

```
Imports System.Data.OleDb
```

```
Public Class SAS_OLE
```

```
'This class is used to interface with SAS as an automation server.
'It should work with SAS for Windows from version 6.12.
'To use from an application use the following syntax:
```

```
'
'Dim SASDemo As New SAS_OLE
```

```
'
'TO SUBMIT CODE:
```

```
'SASDemo.SubmitCode("libname Temp 'C:\Temp';")
```

```
'SASDemo.SubmitCode("data Temp.one;")
```

```
'SASDemo.SubmitCode(" Var1 = 1;")
```

```
'SASDemo.SubmitCode(" output;")
```

```
'SASDemo.SubmitCode("run;")
```

```
'
'TO READ A SAS DATASET:
```

```
'Dim MyDataTable as DataTable
```

```
'MyDataTable = SASDemo.DataSet("C:\Temp","one")
```

```
' Where "C:\Temp" is the directory location of the dataset and
' "one" is the name of the dataset.
```

```
'
'TO SUBMIT A PROGRAM:
```

```
'SASDemo.SubmitProgram("C:\Temp\Program1.sas", True)
```

```
' This will submit a program that has been coded to use the RC
'function to force VB to wait until SAS is finished processing.
```

```
' WARNING: Attempting to use this method without adding SAS code
' to the program being called will result in the application
' getting stuck in an infinite loop.
```

```
'SASDemo.SubmitProgram("C:\Temp\Program1.sas")
```

```
' This will submit a program without any special timing code
added.
```

```
'Variables:
```

```
Protected SAS_Server As New Object '- SAS automation server object
```

```
Protected cnSASData As OleDbConnection '- ADO object for datasets
```

```
Protected Disposed As Boolean
```

```
Private m_DataOnly As Boolean
```

```
Private m_AppPath As String
```

```
Private m_LibName As String
```

```

'Constructors:
Public Sub New(ByVal DataOnly As Boolean)
    'The SAS Server object is created here if DataOnly is set to
    'True. If the SAS object cannot be created an error is
    'throw. The other constructors below call this one and
    'just define additional (or default) properties.
    Try
        Me.Disposed = False
        Me.m_DataOnly = DataOnly
        If Not Me.DataOnly Then
            Try
                Me.SAS_Server = CreateObject("SAS.Application")
                Me.SAS_Server.Wait = True
            Catch ex As Exception
                Dim SASError As New Exception("Error: SAS " & _
                    "is not installed on this system.", ex)
                Throw SASError
            End Try
        End If
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Sub New(ByVal LibName As String, ByVal AppPath As String, _
    ByVal DataOnly As Boolean)
    Me.New(LibName, DataOnly)
    Try
        Me.m_AppPath = AppPath
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Sub New(ByVal LibName As String, ByVal DataOnly As Boolean)
    Me.New(DataOnly)
    Try
        Me.m_LibName = LibName
        Me.OpenLibName()
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Sub New(ByVal AppPath As String, ByVal LibName As String)
    Me.New(AppPath)
    Try
        Me.m_LibName = LibName
        Me.OpenLibName()
    Catch ex As Exception
        Throw ex
    End Try
End Sub

```

```

Public Sub New(ByVal AppPath As String)
    Me.New()
    Try
        Me.m_AppPath = AppPath
        Me.m_LibName = m_AppPath
        Me.OpenLibName()
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Sub New()
    Me.New(False)
End Sub

'Properties:
Property AppPath() As String
    'Path that the application exists in, used as default file
    'location.
    Get
        Try
            If Mid$(m_AppPath, Len(m_AppPath), 1) <> "\" Then
                Me.m_AppPath = Me.m_AppPath & "\"
            End If
            Return Me.m_AppPath
        Catch ex As Exception
            Throw ex
        End Try
    End Get
    Set(ByVal Value As String)
        Try
            Me.m_AppPath = Value
        Catch ex As Exception
            Throw ex
        End Try
    End Set
End Property
Property LibName() As String
    'Directory where sas datasets will be accessed. This property
    'must be set to use the dataset property.
    Get
        Try
            Return Me.m_LibName
        Catch ex As Exception
            Throw ex
        End Try
    End Get
    Set(ByVal Value As String)
        Try
            Me.m_LibName = Value
            Me.OpenLibName()
        Catch ex As Exception
            Throw ex
        End Try
    End Set
End Property

```

```

ReadOnly Property DataOnly() As String
    'Property can only be set when New constructor is called.
    'Determines whether the session will only be used to read
    'SAS datasets (for systems where SAS is not installed, needs
    'SAS Local Data Provider to work).
    Get
        Try
            Return Me.m_DataOnly
        Catch ex As Exception
            Throw ex
        End Try
    End Get
End Property

'Methods:

Public Overridable Sub SubmitCode(ByVal SASCode As String)
    'Submits SAS code to the SAS automation server. Checks
    'for two errors: SAS OLE cannot read strings of more than
    '195 characters and code cannot be submitted when the
    'SAS automation server has not been created yet.
    Try
        If Me.DataOnly <> True Then
            SASCode = SASCode.Trim()
            If SASCode.Length > 196 Then
                Dim CodeError As New Exception("Code must be " & _
                    " less than 196 characters")
                Throw CodeError
            Exit Sub
            Else
                If SASCode <> "" Then
                    Me.SAS_Server.Submit(SASCode)
                End If
            End If
        Else
            Dim CodeError As New Exception("SAS_OLE is not set" & _
                " up to work with SAS code. Run with DataOnly" & _
                " option set to True.")
            Throw CodeError
            Exit Sub
        End If
    Catch ex As Exception
        Throw ex
    End Try
End Sub

Public Sub SubmitProgram(ByVal FileName As String)
    'Submits a SAS program without any timing added. If called
    'multiple times program will not wait for the previous
    'program to finish before executing.
    Try
        Me.SubmitProgram(FileName, False)
    Catch ex As Exception
        Throw ex
    End Try
End Sub

```

```

Public Sub SubmitProgram(ByVal FileName As String, _
                        ByVal WaitForProgramToFinish As Boolean)

    'Submits a SAS program with the option of using timing
    'to force the system to wait for the program to finish
    'return moving on to the next program.

    'WARNING: If you use the option WaitForProgramToFinish = True
    'then you must include special code from within your SAS
    'program to set the arc value in SAS to 1 when the program is
    'done running.

    Try
        FileName = AppPath & FileName
        FileName = FileName.Trim()
        If FileName <> "" Then
            If WaitForProgramToFinish Then
                Me.SubmitCode("data _null_;")
                Me.SubmitCode("error=setrc('Start',0.0);")
                Me.SubmitCode("put error;")
                Me.SubmitCode("run;")
                Me.SubmitCode("%include '" & FileName & "';")
                Do Until SAS_Server.rc = 1
                    Loop
                Me.SubmitCode("data _null_;")
                Me.SubmitCode("error=setrc('Start',0.0);")
                Me.SubmitCode("put error;")
                Me.SubmitCode("run;")
            Else
                Me.SubmitCode("%include '" & FileName & "';")
            End If
        Else
            Dim CodeError As New Exception("SAS_OLE.Submit" & _
                "Program: Attempted to submit a SAS program with" & _
                "a blank filename")
            Throw CodeError
            Exit Sub
        End If
    Catch ex As Exception
        Throw ex
    End Try
End Sub

```

```
Protected Overridable Sub OpenLibName()  
    'Opens a data connection to the directory where datasets are  
    'located. Throws an error is the libname has not been  
    'defined.  
    Try  
        Me.CloseLibName()  
        If Me.LibName.Trim <> "" Then  
            Me.cnSASData = New OleDbConnection("Provider=" & _  
                "SAS.LocalProvider.1;Data Source=" & Me.LibName)  
            Me.cnSASData.Open()  
        Else  
            Dim ADOError As New Exception("SAS_OLE. " & _  
                "OpenLibName: Libname is not defined, system " & _  
                "cannot make connection to dataset directory.")  
            Throw ADOError  
        End If  
    Catch ex As Exception  
        Throw ex  
    End Try  
End Sub  
Protected Overridable Sub CloseLibName()  
    'Closes the data connection if it is open.  
    Try  
        If Not Me.cnSASData Is Nothing Then  
            If Me.cnSASData.State = ConnectionState.Open Then  
                Me.cnSASData.Close()  
            End If  
        End If  
    Catch ex As Exception  
        Throw ex  
    End Try  
End Sub
```

```
Public Overridable Function DataSet(ByVal DataSetName As String) _
    As DataTable
    'Function returns an ADO datatable that is populated by a sas
    'dataset. Libname would have had to been set for this method to
    'work.
    Try
        'open data connection if it is closed:
        If Not Me.cnSASData Is Nothing Then
            If Me.cnSASData.State = ConnectionState.Closed Then
                Me.OpenLibName()
            End If
        Else
            Me.OpenLibName()
        End If

        'create command object:
        Dim cmd As OleDb.OleDbCommand = Me.cnSASData.CreateCommand
        cmd.CommandType = CommandType.TableDirect
        cmd.CommandText = DataSetName

        'create dataset, dataadapter and then fill with sas dataset
        'rows.
        Dim da As OleDb.OleDbDataAdapter = _
            New OleDb.OleDbDataAdapter(cmd)
        Dim ds As New DataSet
        Dim dt As New DataTable
        da.Fill(ds, DataSetName)

        'if successful return a copy of the datatable to the client
        'application:
        If ds Is Nothing Then
            Return Nothing
        Else
            If ds.Tables(0).Rows.Count = 0 Then
                Return Nothing
            Else
                dt = ds.Tables(0).Copy
                Return dt
            End If
        End If

        Catch ex As Exception
            Throw ex
        End Try
    End Function
```

```

Public Overridable Function DataSet(ByVal LibName As String, _
                                   ByVal DataSetName As String) As DataTable
    'Sets the libname and then returns the specified dataset.
    Try
        Me.LibName = LibName
        Return Me.DataSet(DataSetName)
    Catch ex As Exception
        Throw ex
    End Try
End Function
Public Overridable Sub StartLog(ByVal FileName As String, _
                                ByVal ClearFirst As Boolean)
    'Starts a printing the SAS log out to a specified FileName.
    'SaveLog must be called to finish writing out the file.
    'Throws exception if FileName is blank.
    Try
        If FileName.Trim <> "" Then
            Me.SubmitCode("FileName x '" & FileName & "';")
            If ClearFirst Then
                Me.SubmitCode("proc printto log=x new;")
            Else
                Me.SubmitCode("proc printto log=x;")
            End If
            Me.SubmitCode("run;")
        Else
            Dim LogError As New Exception("SAS_OLE." & _
                "StartLog: FileName is blank, log file not" & _
                " created.")
            Throw LogError
        End If
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Overridable Sub StartLog(ByVal ClearFirst As Boolean)
    'Starts log file in default AppPath location if not filepath
    'is specified.
    Try
        Dim FileName As String
        If Me.AppPath.Trim <> "" Then
            FileName = Me.AppPath & "SAS_Log.txt"
        Else
            FileName = ""
        End If
        Me.StartLog(FileName, ClearFirst)
    Catch ex As Exception
        Throw ex
    End Try
End Sub

```

```

Public Overridable Sub SaveLog()
    'Writes out SAS log, StartLog must be called prior to this
    'method.
    Try
        Me.SubmitCode("proc printto log=log print=print;")
        Me.SubmitCode("run;")
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Overridable Sub StartList(ByVal FileName As String, _
                                ByVal ClearFirst As Boolean)
    'Starts a printing the SAS output window out to a specified
    'FileName. SaveLog must be called to finish writing out
    'the file. Throws exception if FileName is blank.
    Try
        If FileName.Trim <> "" Then
            Me.SubmitCode("FileName x2 '" & FileName & "';")
            If ClearFirst Then
                Me.SubmitCode("proc printto print=x2 new;")
            Else
                Me.SubmitCode("proc printto print=x2;")
            End If
            Me.SubmitCode("run;")
        Else
            Dim ListError As New Exception("SAS_OLE." & _
                "StartLog: FileName is blank, list file not" & _
                " created.")
            Throw ListError
        End If
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Overridable Sub StartList(ByVal ClearFirst As Boolean)
    'Starts output file in default AppPath location if not filepath
    'is specified.
    Try
        Dim FileName As String
        If Me.AppPath.Trim <> "" Then
            FileName = Me.AppPath & "SAS_List.txt"
        Else
            FileName = ""
        End If
        Me.StartList(FileName, ClearFirst)
    Catch ex As Exception
        Throw ex
    End Try
End Sub

```

```

Public Overridable Sub SaveList()
    'Writes out SAS output, StartList must be called prior to this
    'method.
    Try
        Me.SubmitCode("proc printto print=print;")
        Me.SubmitCode("run;")
    Catch ex As Exception
        Throw ex
    End Try
End Sub
Public Overridable WriteOnly Property Visible() As Boolean
    'Setting this property to True or False will
    'determine if the SAS server can be viewed by
    'the user.
    Set(ByVal Value As Boolean)
        Try
            If Me.DataOnly = False Then
                Me.SAS_Server.Visible = Value
            End If
            Catch ex As Exception
                Throw ex
            End Try
        End Set
    End Property

    'Other class methods:
    Public Overrides Function ToString() As String
        'Defines default ToString behavior.
        Try
            If Me.DataOnly Then
                Return "SAS Local Data Provider: " & Me.LibName
            Else
                Return "SAS OLE Automation: " & Me.AppPath
            End If
            Catch ex As Exception
                Throw ex
            End Try
        End Function
    Protected Overrides Sub Finalize()
        'Needed to work with garbage collector to
        'dispose of SAS objects.
        Try
            If Not Me.Disposed Then
                MyBase.Finalize()
                Me.Dispose()
            End If
        Catch
        End Try
    End Sub

```

```
Public Overridable Sub Dispose()  
    Try  
        'Disposes SAS automation server.  
        Me.CloseLibName()  
        Me.SAS_Server.Quit()  
        Me.SAS_Server.SAS = Nothing  
        Me.Disposed = True  
    Catch  
    End Try  
End Sub  
  
End Class
```