

Building Windows Front ends to SAS® Software

Katie Essam, Amadeus Software Limited



ABSTRACT

Version 8 of SAS Software introduced far greater interoperability with other software provider's technology. Combining this with the exciting functionality available in Microsoft's .NET® framework provides us with the perfect partnership for creating easy-to-use, highly functional, attractive and effortlessly deployable Graphical User Interfaces (GUI) that utilise the power of SAS Software.

This paper will discuss the methods available for communication between a Visual Basic .NET GUI and SAS Software, including ADO .NET, SAS Workspaces and the SAS COM components.

INTRODUCTION

Prior to Version 8 of SAS Software application development was limited to SAS/AF® Software, a product within SAS Software that enabled the creation of User Interfaces. With the introduction of SAS Integration Technologies within Version 8, SAS Institute have made the functionality of SAS Software available to developers working in other languages, such as Java, C#, C++ and Visual Basic.

This paper aims to provide an introduction to some of the methods that we can use to access SAS Software from within as Visual Basic .NET application, some familiarity with SAS Software and the Visual Basic Language would be advantageous but is not assumed.

The examples in this paper should provide an introduction to some of the techniques that can be used to execute SAS programs and access SAS Data Sources. However, I would like to add a caveat that the code provided works but there may be more efficient ways of accessing SAS from VB.NET.

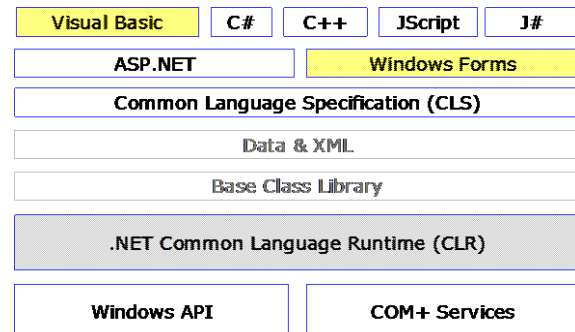
WHAT IS .NET?

Microsoft's .NET provides an all new view of the windows operating system. It provides developers with a single approach to building rich client or web client applications.

The .NET Framework is comprised of hundreds of classes that provide the functionality of Windows. It also provides a Common Language Runtime (CLR) so that all .NET programming languages can be used to develop applications and are equal in power.

Today it is possible to write applications for Apple Mac, for more information see the Microsoft Corporation website. In the future the ability to develop applications using .NET that run on UNIX or any other operating system should be possible.

The diagram below gives an overview of .NET.



Visual Basic .NET is the latest version of the visual basic language and is now a fully object orientated programming language, directly comparable to other Visual Studio languages, such as Visual C++ and Visual C#.

From the developers point of view Visual Basic (VB) provides us with a rapid application development tool (RAD), highly reusable code and a large pool of resources. From a users point of view visual basic can be used to provide form based Windows applications with rich functionality and a familiar look and feel.

Several different types of application can be developed in VB including Console Applications, Windows Form Applications (or Rich Client), Web Applications and Web Services

This paper focuses on using a Windows Form Application.

SAS SOFTWARE'S INTEROPERABILITY

Prior to Version 8 of SAS Software applications that wanted to use SAS Data Sources were built using SAS/AF® Software, proprietary software with it's own application development language called SCL (Screen Control Language in V6 or SAS Component Language in V8). Version 8 included a major overhaul of the SCL language that provided 'dot' notation for referencing objects and their methods in a manner much more inline with other languages. However, developing applications using SAS/AF required a skill set not common in organisations.

Version 8 introduced SAS Integration Technologies, which lays the foundation to enable applications to seamlessly integrate with SAS using industry standard tools. SAS Integration Technologies provides the middleware to allow application developers, working in a number of different languages, to quickly create applications that harness SAS Software's functionality.

The following industry standards are supported by SAS Integration Technologies:

Directory Technology

- LDAP (Lightweight Directory Access Protocol)
Used for accessing a distributed store of information.

Distributed object models

- Microsoft's COM/DCOM
The Component Object Model (COM) allows objects in different object spaces to 'talk' to each other by calling each

other's methods. DCOM (Distributed COM) allows components to communicate with each other over networks

NOTE SAS Integration technologies provides a set of COM objects known as the Integrated Object Model (IOM). This is the most commonly used tool for Visual Basic applications that access SAS.

- CORBA (Common Object Request Broker Architecture)
CORBA is a vendor independent infrastructure to provide communication between applications over networks

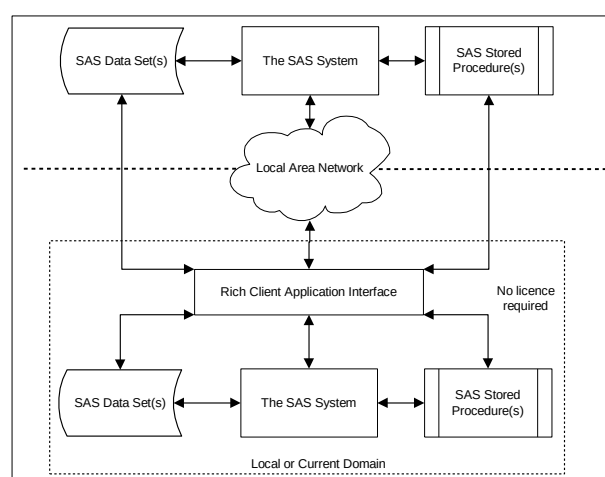
Message orientated middleware

- IBM's MQSeries and Microsoft's MSMQ
Provides standard for allowing requests and response to be sent and received as 'messages'

INTEGRATED OBJECT MODEL

The IOM is an object model in SAS Integration Technologies that provides a set of objects that interface to Base SAS Software features, including procedures, data, files, results and formatting. The IOM enables application developers to develop client applications, written in a number of industry standard languages, to communicate with these features via an IOM server. Client applications can connect to an IOM server through a variety of industry standard communication protocols, including CORBA, JDBC and COM. The IOM objects mainly handle two tasks: to submit Language programs to SAS and to retrieve information and data back from SAS.

The IOM interfaces are available within Base SAS Software for Windows. However, the way in which they can be used is dependent on the SAS Software products licenced. If the Client application is to communicate with SAS using local COM (the application and SAS are on the same machine) then no additional licences are required. NOTE that some functionality is restricted when using the Local Provider such as SQL processing and results packages. However, to communicate with an IOM server across the network SAS Integration Technologies must be licenced to allow communication via DCOM and TCP/IP (using the IOM Bridge for COM). These concepts are illustrated in the following diagram.

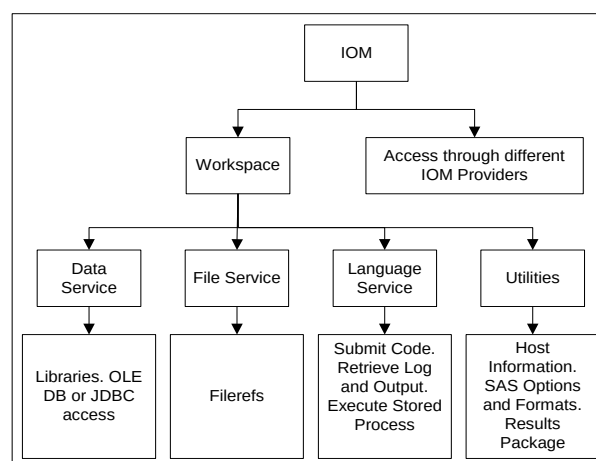


The principle IOM interfaces are:

- Workspace
Allows SAS Sessions to be stopped and started, both remotely and locally
- LanguageService

Provides methods for submitting SAS code and retrieving the log and textual output. Can be used to raise events on DATA and PROC steps starting and stopping, ERROR's in code and at the end of SUBMITTED code. Enables Stored Processes to be called.

- DataService
Allows the creation, removal and access of SAS Libref's within the current Workspace
- FileService
Provides an interface to SAS Fileref's
- Utilities
Provides interfaces to options, formats and more sophisticated output via the ResultsPackage.



The full object hierarchy can be found at

<http://support.sas.com/rnd/itech/doc/dist-obj/iom.html>

The root object of the IOM Hierarchy is the Workspace. The Workspace can be created via the Workspace Manager, which establishes a connection with SAS. The Workspace Manager is not necessarily required but makes scaling the application to use multiple Workspaces easy. To establish a connection to SAS using the workspace the following code can be used in VB.NET:

```
Dim obWsMgr As New SASWorkspaceManager.WorkspaceManager()
Dim obWS As SAS.Workspace
Dim errString As String
obWS=obWsMgr.Workspaces.CreateWorkspaceByServer("Session", _
    SASWorkspaceManager.Visibility.VisibilityNone, _
    Nothing, "", "", errString)
```

This is covered in more detail in the RUNNING SAS PROGRAMS section.

As SAS Integration Technologies depends on the SAS language to perform data analysis the LanguageService provides the methods for submitting SAS Code to the established Workspace. This is covered in more detail in the RUNNING SAS PROGRAMS section.

IOM DATA PROVIDERS

There are three SAS Data Providers that allow direct connections to SAS data sources. These are listed below, along with their key features:

- SAS Local Data Provider
 - Allows manipulation of data sets in your windows environment
 - Does not support SQL command processing

- Single User update
- SAS/SHARE Data Provider
 - Allows manipulation of data sets and views in a SAS/SHARE server.
 - Multi User update
 - Supports SQL processing
- SAS IOM Data Provider
 - Allows manipulation of data sets and views in an Integration Technologies object server.
 - Supports SQL processing
 - Multi user support
 - Can be used on any platform that the IOM supports

All of the providers:

- Have basic OLE DB schema rowsets, which enable consumers to obtain metadata about the data source they are using.
- Have integrated SAS Formatting services (NOTE the Local provider cannot use user defined formats)

For the purposes of this paper I will focus on using the Local Data Provider as this doesn't require anything more than a Base SAS licence. Detailed information about using each of the providers can be found in SAS Institute's documentation.

WHAT IS OLE DB?

OLE DB is a Microsoft specification that contains a set of interfaces for data access. OLE DB is expected to become the dominant method of data access as it provides universal access to data on the Windows platform and so is not dependent on proprietary software. The IOM Data Provider is an OLE DB provider that enables communication with the IOM server.

GETTING SAS TO COMMUNICATE WITH VISUAL BASIC .NET

.NET can communicate with SAS by making use of any of COM (or DCOM), OLE/DB, ADO.NET or ODBC.

COM(or DCOM) protocols are used indirectly when using the SAS IOM, discussed in the RUNNING SAS PROGRAMS section.

The other Industry standard protocols can be used to access SAS data directly, as discussed in the ACCESSING DATA section.

RUNNING SAS PROGRAMS

There are a number of options available that enable us to run SAS programs via the IOM. These include: Stored Processes, simple file includes or submitting code created within the application. Examples of each of these will be shown in this paper.

However, we first have to setup Visual Basic .NET so that it can communicate with SAS. To do this we first need to install SAS Institute's Integrated Technologies. This is available for PC's and a self-extracting executable is available to download from SAS Institute's Web Site. This executable will install the necessary components on your PC.

Once the Integration Technologies Components have been installed we need to add a reference to them within the VB

.NET project. The two components to add a reference to are:

1. SAS Integrated Object Model (IOM) (SAS System Version 8.2) Type Library – Type Lib Version 1.1
2. SASWorkspaceManager Type Library 1.1

Once these references have been added we can make use of these components in our code.

STARTING SAS

To start a SAS session we need to create an instance of the SAS Workspace. As this code is likely to be reused many times it is good practice to create a standalone function or sub routine. First we create an instance of the SAS Workspace Manager, which controls all instances of a SAS Workspace, and a SAS Workspace. By declaring these as public we make them available to all functions and sub routines in the application.

```
Public obWsMgr As New SASWorkspaceManager.WorkspaceManager()
Public obWS As SAS.Workspace
```

To create a Workspace, or SAS session, we use the Workspace Manager, obWsMgr, to create a new workspace.

```
Public Function StartSAS()
    Dim errString As String
    obws = obWsMgr.Workspaces.CreateWorkspaceByServer( _
        "My workspace", _
        SASWorkspaceManager.Visibility.VisibilityNone, _
        Nothing, "", "", errString)
End Function
```

After calling this function a SAS Workspace, in this case called obws, is ready for use.

Notice the _ at the end of the lines of code. This is used by VB to indicate that the code continues on the following line. This will be used as necessary in the following examples.

The Workspace created by this function will keep running until we stop it, so we need a corresponding function to close SAS.

STOPPING SAS

This function will stop the SAS Session. The close method closes the SAS session. In Visual Basic objects persist, or stay in existence, until they are removed. By setting the Workspace to Nothing we remove it from use

```
Public Function EndSAS()
    obws.Close()
    obws = Nothing
End Function
```

INCLUDING A SAS PROGRAM - %INCLUDE

One of the simplest ways of running a SAS program is to use a %include statement to specify the name of the SAS program that we want to run.

This example assumes a form has been created with a button called Submit and a TextBox called filename. A String is created that contains the %include statement and the full path name of the SAS Program that we want to run (filename.text). The StartSAS function is called to create a SAS Workspace, or session. The String that contains the SAS code is then passed to a function called SubmitCode. Once the code has been submitted the SAS Workspace is closed by calling the EndSAS function.

```
Private Sub Submit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles _
```

```

        Submit.Click
Dim SASCode As String
SASCode = "%include '" & fileName.Text & "';"
StartSAS ()
SubmitCode (SASCode)
EndSAS ()
End Sub

```

To submit the code an instance of the SAS Language Service is created. This enables us to pass the SAS code to the SAS Workspace for processing, using the Submit method. Again, once the object is finished with, in this case the obLS, it is cleared by setting it to Nothing.

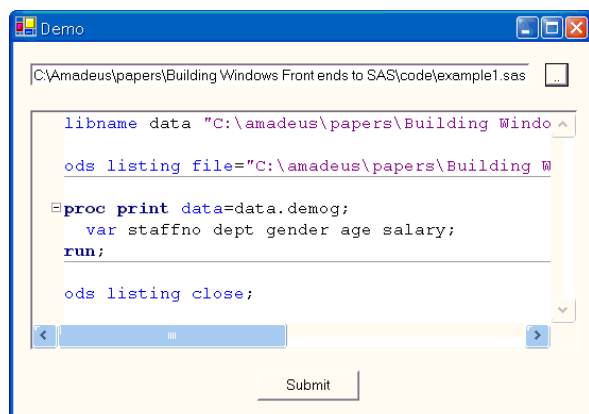
```

Private Function SubmitCode(ByVal SASCode As String) As Boolean
Dim obLS As SAS.LanguageService = obws.LanguageService
obLS.Submit (SASCode)
obLS = Nothing
End Function

```

The SAS Language Service can also be used to retrieve the SAS Log and SAS Output, by using the FlushLogLines and FlushListLines respectively.

This idea could be extended to create a screen that enables the users to write the own code and submit it, as shown below.



Instead of creating a String that holds the SAS Code as in the previous example, we can simply pass the text of the editor object to the SubmitCode function.

```

Private Sub Submit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles _
    Submit.Click
StartSAS ()
SubmitCode (SASEditor.Text)
EndSAS ()
End Sub

```

INCLUDING A SAS PROGRAM – USING AN ARRAY

An alternative method to passing a single String, which contains the SAS Program is to create an String Array that contains a line of code in each element in the Array. An example of this is shown below.

```

Private Sub Submit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles _
    Submit.Click
Dim SASCode (6) As String
SASCode(0) = "libname data ""C:\temp"";"
SASCode(1) = "ods listing file=""C:\temp\output.txt"";"
SASCode(2) = "proc print data=data.demog;"
SASCode(3) = "var staffno dept gender age salary;"
SASCode(4) = "run;"
SASCode(5) = "ods listing close;"

```

```

StartSAS ()
SubmitCode (SASCode)
EndSAS ()
End Sub

```

Of course, in the above example the SAS code is hard coded into the Visual Basic application. However, we could read a SAS Program in, assigning each line in the program to an element in the array.

When submitting an Array, as opposed to a single String, using the Language Service the code changes slightly to use the SubmitLines method in place of the Submit method. For example,

```
obLS.SubmitLines (SASCode)
```

STORED PROCESS

Perhaps the preferred method for storing and executing SAS Programs is to treat each SAS Program as a Stored Process. A Stored Process in SAS terms is simply a .sas file which has a special comment in the header of:

```
*ProcessBody;
```

Stored Processes can be thought of as macros within Visual Basic. The advantage of using Stored Processes is that the application logic is completely separate from the business logic contained within the Stored Process. Therefore, any changes to the business logic (our SAS code) do not impact the application logic.

To use a Stored Process in SAS we need to specify the Repository of Stored Processes (or the directory containing the SAS files), the name of the SAS program that we want to run and any required macro variables. If no macro variables are required then a null value can be passed, as in the example below.

```

Private Function SubmitCode(ByVal dir As String, _
    ByVal program As String) As Boolean
Dim obLS As SAS.LanguageService = obws.LanguageService
Dim obSP As SAS.StoredProcessService = _
    obLS.StoredProcessService

obSP.Repository = "file:" & dir
obSP.Execute (program, "")

obLS = Nothing
obSP = Nothing
End Function

```

If macro variable values are required the macro variables should be defined as Variable=Value. To do this we could use code similar to the following:

```

cmd = "title='This is the title'"
obSP.Execute (program, cmd)

```

EVENTS

The Language Service also provides the functionality to dynamically capture events as they happen in an executing SAS program. Events that the Language Service can capture include when a data or proc step stops or starts and when the SAS program stops and starts execution and also when any errors occur.

To make these events accessible, we need to declare the Language Service using WithEvents, as shown.

```
Dim WithEvents obLS As SAS.LanguageService
```

We then submit code in the same manner as the previous example.

```
Private Function SubmitCode(ByVal dir As String, _
    ByVal program As String) As Boolean
    obLS = obWS.LanguageService

    (continues as before)

End Function
```

However, for this to work with the Local IOM in Version 8.2 of SAS Software you may need to modify the DCOMCNFG settings, by executing the command DCOMCNFG at the DOS prompt. In the Applications tab, select the properties for SAS: IOM DCOM Servers. The Authentication Level should be set to "Connect".

In the Default Properties tab, the Default Authentication Level: should be "(None)" and the Default Impersonation Level: should be set to "Impersonate". NOTE: You may need to reboot for these settings to take effect.

Once these settings have been changed we can add code to capture events within our SAS programs. In the example, below a message is written to a StatusBar when a data step completes.

```
Private Sub obLS_DatastepComplete() Handles _
    obLS.DatastepComplete
    StatusBar.Panels(0).Text = "DATA Step has finished"
End Sub
```



ACCESSING DATA

Visual Basic.NET uses ADO.NET (ActiveX Data Objects for the .NET framework) as its primary data access and manipulation tool. ADO.NET makes use of OLE DB interfaces to provide standard access methods to a variety of data sources, including Microsoft SQL Server and Access, as well as SAS data formats. Although the concepts for accessing the different data sources are the same the manner in which we implement a connection to SAS differs slightly from the methods used by Microsoft. The facility to connect to a SAS Data Set provides us with the ability to display information about a SAS Data Set, such as its columns and labels, as well as a SAS Data Set itself within our VB.NET applications.

There are two objects that can be used to access SAS Data Sets: these are a DataAdaptor or a DataReader. Our choice of access method is dependent on the task that is required. The DataReader provides sequential, read only access to a SAS Data Set. The DataAdaptor provides read-write, multi directional access, along with the ability to perform SQL queries against the SAS Data Source (NOTE: this is dependent on the IOM Provider in use, which is dependent on the SAS products that are licenced)

For the following examples the following connection object is created.

```
Public cn As New OleDb.OleDbConnection()
```

The OleDbConnection provides the interface that allows us to establish a connection via an IOM provider to a SAS Data Source.

USING THE OLEDB SCHEMA

The following example assumes that a VB.NET DataTable has been defined. The DataTable is a key component of the ADO.NET Data Library and it represents one table of relational in-memory data. The created DataTable is local to our VB.NET application, which enables us to display information in our application without maintaining a connection to the original data source. The speed of our application can also be greatly enhanced as it uses a local copy of the data rather than having to query the database continuously. The code below creates a public instance of a DataTable, so that it is accessible to all classes within our application.

```
Public schema As DataTable
```

The following example connects to a SAS Data Set and accesses it's header portion to populate a ListBox with the Data Set's column names, as shown in the figure below.

To establish a connection to a SAS Data Set we need to provide the name of the IOM Provider that we are using and the directory where our SAS Data Set resides (SASLibrary in the example below). Once the connection has been opened we can use the GetOleDbSchemaTable method, which enables us to retrieve schema information about a Data Source. In the example below, we are asking for the columns schema of the SAS Data Set, the name of which is stored in the String SASDataSet

```
Public Function Make_Connection(ByVal SASLib As String, _
    ByVal SASDataSet As String) As Boolean
    cn.ConnectionString = _
        "Provider=sas.LocalProvider;Data Source=" & SASLibrary
    cn.Open()
    schema = cn.GetOleDbSchemaTable(OleDbSchemaGuid.Columns, _
        New String() {"", "", SASDataSet})
End Function
```

To make a connection, we pass the path of the SAS Data Set and the SAS Data Set name to the above function. Once the connection has been made, we can access the resulting DataTable. In this example, we loop round the rows of the DataTable, retrieving the value of the column "Column_Name" from the metadata returned by the function.

```
Make_Connection(SASLib, SASDataSet)
Dim i As Integer
For i = 0 To schema.Rows.Count - 1
    ListBox.Items.Add(schema.Rows(i).Item("Column_Name"))
Next
Close_Connection()
```

Once we have finished with the DataTable, we close the connection and clear the DataTable.

```
Public Sub Close_Connection()
    If cn.State Then cn.Close()
    schema.Clear()
End Sub
```

USING A DATAADAPTOR

In addition to being able to retrieve metadata about a SAS Data Set we may want to display the data itself, perhaps in a DataGrid as shown below.

	age	gender	height	weight	status
▶	22	M	64	60	S
	26	F	87	74	M
	29	M	55	62	D
	24	M	84	74	M
	23	F	84	88	P
	30	M	87	76	S
	34	F	78	72	M
	38	F	69	56	M
	56	F	52	60	M
	52	F	66	69	M
	60	F	88.5	71	M
	56	F	61	61	M
	46	F	63	76	W
	65	F	87	81	M
	23	F	78	80	S
◀	28	M	67	82	

Depending on the task we want to achieve we could use either a DataReader or a DataAdapter to access the data set, as discussed earlier. This example demonstrates using the DataAdapter.

Using the same connection as defined in the previous example we specify the name of the SAS IOM Provider and the directory containing the SAS Data Set and then open the connection. We then create an OleDbCommand, which will be used, in conjunction with the DataAdapter, to fill an ADO.NET DataSet. A DataAdapter can be thought of as a bridge between a ADO.NET DataSet and an external data source. The Fill method is used to retrieve data from an external source into a DataSet.

The **DataSet** is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source. It can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The **DataSet** represents a complete set of data including related tables, constraints, and relationships among the tables

Source: .NET Framework Developer's Guide

We can think of a DataSet as being a collection of DataTables.

```
Private Sub GetData()
    cn.ConnectionString = _
        "Provider=sas.LocalProvider;Data Source" & SASLib
    cn.Open()

    Dim cmd As OleDb.OleDbCommand = cn.CreateCommand()
    cmd.CommandType = CommandType.TableDirect
    cmd.CommandText = "demog"
    Dim oleDA As OleDbDataAdapter = New _
        OleDbDataAdapter(cmd)

    Dim ds As DataSet = New DataSet()
    oleDA.Fill(ds, "demog")
    DataGrid.DataSource = ds
    DataGrid.DataMember = "demog"
End Sub
```

The CommandType specifies how we want to access the external data source, in this case using the TableDirect type.

The CommandText in this example is the name of the SAS Data Set that we want to access. This command is then used in the creation of the OleDbDataAdapter to create the link to the external data source. We then use the Fill method to transfer the data from the SAS Data Set, called demog, to the ADO.NET DataSet, called ds. Once the DataSet is populated it can be linked to a DataGrid by specifying the DataMember, which is a DataTable, in this case demog, within the DataSet ds, the DataSource.

NOTE: This replaces the classic ADO connection which used an ADO recordset

CONCLUSION

By introducing SAS Integration Technologies, SAS have provided us with the mechanism to utilise the functionality of SAS Software as the back end to User Interfaces, which can be written in a variety of industry standard languages. This paper introduced some of the key functions that a developer may wish to incorporate into their applications.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Katie Essam
 Amadeus Software Ltd
 Orchard Farm
 Witney Lane
 Leafield, United Kingdom, OX29 9PG
 Phone: +44 (0) 1993 878287
 Fax: +44 (0) 1993 878042
 Email: katie.essam@amadeus.co.uk
 Web: www.amadeus.co.uk

TRADEMARKS

SAS and all other SAS Institute Inc. product or service names are the registered trademarks or trademarks of SAS Institute Inc.

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

COPYRIGHT

No part of this material may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Amadeus Software Ltd. © Amadeus Software. May 2003. All rights reserved.

REFERENCES

SPECIFIC

<i>Visual Basic .NET and The SAS System</i>	David Shannon, Amadeus Software Seminar, 2003
<i>Developing an Open Client in Visual Basic</i>	Daniel Jahn, SAS Institute Inc
<i>What is SAS Integration Technologies?</i>	Vijay Rajandram, Amadeus Software Seminar, 2002
<i>ADO and OLE DB Cookbook for Release 8.2</i>	SAS Institute Inc
<i>Using IOM and Visual Basic in SAS® Program Development</i>	Greg Silva, Biogen Inc., SUGI 28

GENERAL

<http://support.sas.com/rnd>