Blaise API, a practical application.

Pauline Davis, ONS
Andrew Tollington, ONS

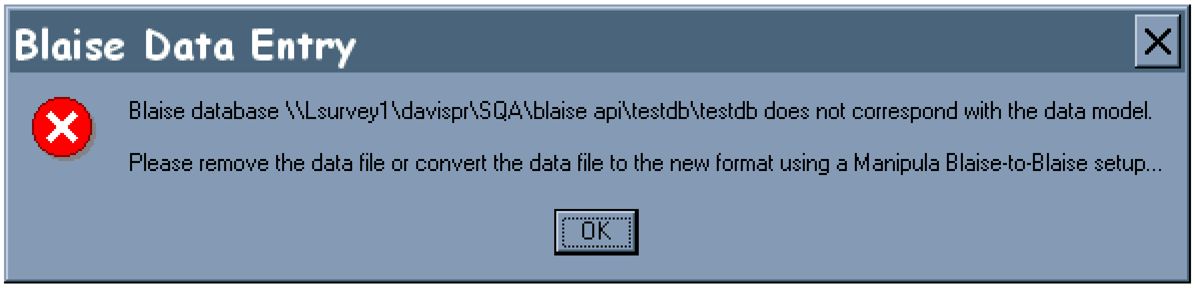
Keywords
Application Programming Interface (API)
Component Object Model (COM)
Blaise [meta] data
Windows

Introduction
The arrival of an API for Blaise will enable developers to design and build powerful Windows applications that not only harness the core Blaise engine but extend and achieve functionality that would otherwise be complex - if not impossible, time-consuming and costly - using traditional Manipula/Cameleon programming techniques.

The Blaise API provides total access to your data and meta-data in a structured, object based manner. In exposing a COM interface it can be implemented with any supporting language of your choice from JavaScript to COBOL.

The scope of the practical application of this technology is seemingly limitless and this paper will discuss the design and build of an example application that deals with a typical problem and demonstrates some of these points. It is not intended to delve deeply into COM programming or every facet of the Blaise Object Model but to provide an introduction to the many possibilities now available for the Windows Developer and their value to the Blaise programmer.

The example application - data file incompatibility problems
Blaise checks to see whether the current data description is compatible with the current data file. If it isn't, when you try to run the Data Entry Program (DEP) you get an incompatibility error message similar to the one given below.



Blaise database \\Lsurvey1\davispr\SQA\blaise api\testdb\testdb does not correspond with the data model.

Please remove the data file or convert the data file to the new format using a Manipula Blaise-to-Blaise setup...

Incompatibility between the data model and the data file can occur for a number of reasons, some of which are when you:

- Change the number of CHECKS or SIGNALS
- Change the attributes of a field
- Change the number of fields
- Change the order of fields in the FIELDS section
- Change the valid range of fields
- Change the number of choices in an enumerated type
- Change the primary key definition
- Change the secondary field definition

Incompatibility can occur even when making small changes. Whilst developing a questionnaire data file incompatibility is inevitable. If the data file becomes incompatible whilst you are developing it is easy simply to delete the database and start again, but if the questionnaire is already in the field this can prove much more difficult. This is usually solved by sending out a batch file/Visual Basic Script(VBS) with the new version of the questionnaire, that updates the dataset. The majority of the time you can easily create Manipula script to convert the data from one form to another, but if for example a field type has changed this means producing long script which converts each block and then in the block that has changed, each field.

Summary of data file incompatibility problems

1. Sending out incompatible datamodels, sometimes unknowingly and then having to create a quick fix.

2. Having to write long Manipula script to convert from one model to another e.g where field types have changed.

3. Not being able to quickly find out the differences between two data models if you do not have the original code.

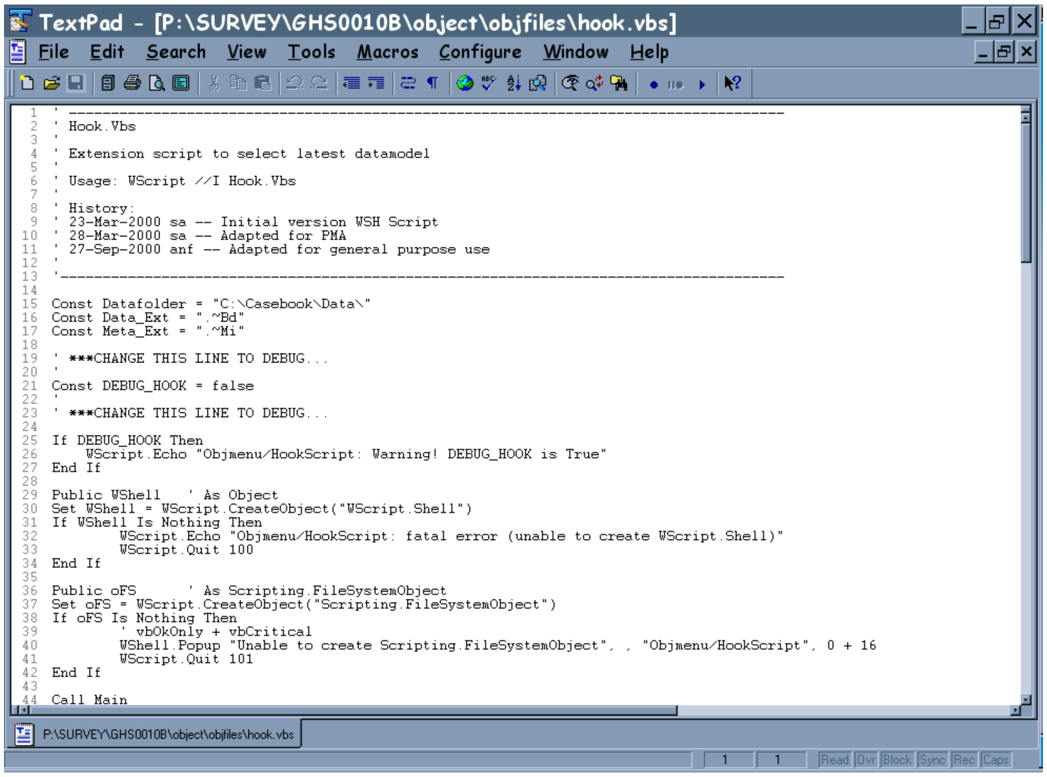
Current solutions

1. Sending out a hook with the new version of the questionnaire which runs a Manipula script to convert the data from one version of the questionnaire to another (fig 1.)

2. Writing long Manipula script to convert the current data file to the current data description. (fig 2.)

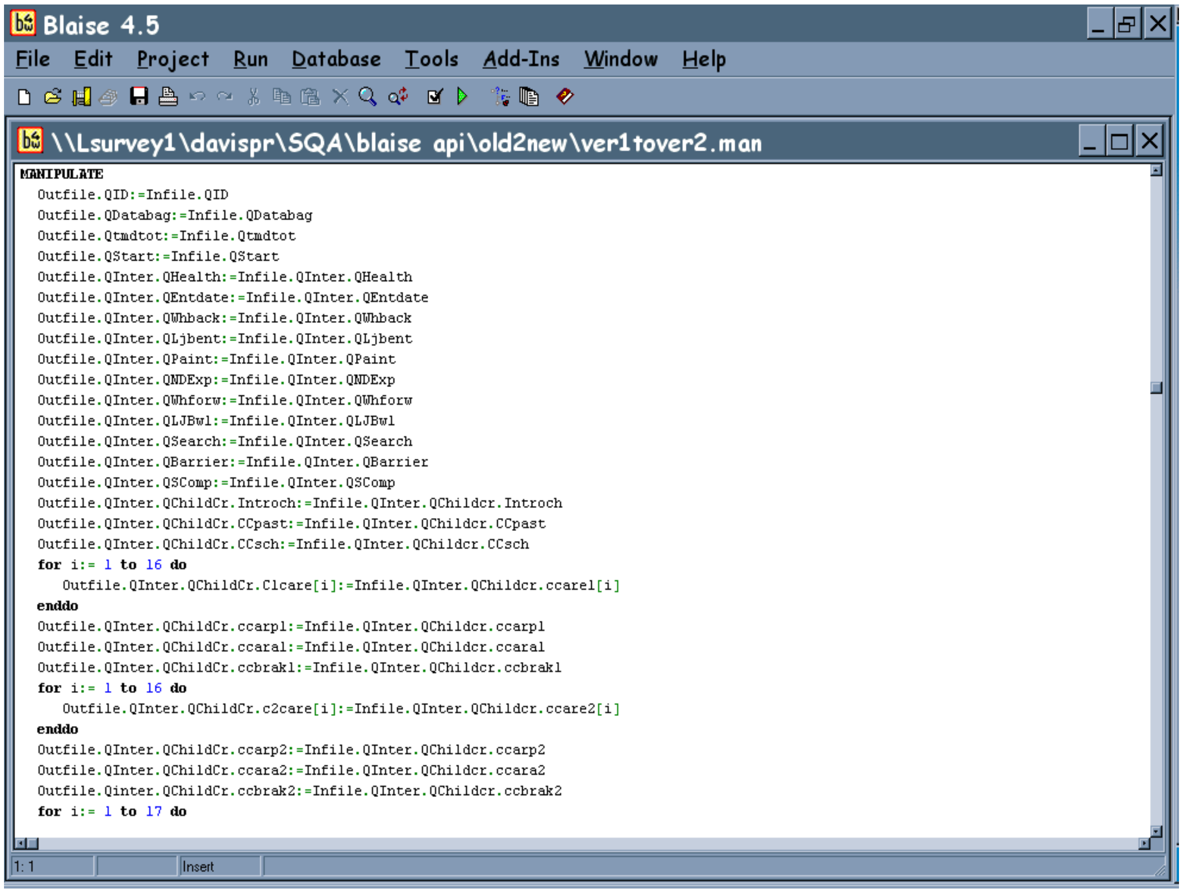
2. Using the Cameleon, Manipula and Maniplus tools to identify the differences between two datamodels.

All of these solutions are time consuming and prone to error. A better solution would be one that combines access to metadata and data at the same time, that could identify an incompatible datamodel with a questionnaire of the same name already in the field, let you know the differences between two datamodels and where appropriate write out the Manipula script. This is all possible using the Blaise API.


TextPad - [P:\SURVEY\GHS0010B\object\objfiles\hook.vbs]
File Edit Search View Tools Macros Configure Window Help
P:\SURVEY\GHS0010B\object\objfiles\hook.vbs
Read Ovr Block Sync Rec Caps

```
' ------------------------------------------------------------------------------------
' Hook.Vbs
'
' Extension script to select latest datamodel
'
' Usage: WScript //I Hook.Vbs
'
' History:
' 23-Mar-2000 sa -- Initial version WSH Script
' 28-Mar-2000 sa -- Adapted for PMA
' 27-Sep-2000 anf -- Adapted for general purpose use
'
'-------------------------------------------------------------------------------------

Const Datafolder = "C:\Casebook\Data\"
Const Data_Ext = ".~Bd"
Const Meta_Ext = ".~Mi"

' ***CHANGE THIS LINE TO DEBUG...
'
Const DEBUG_HOOK = false
'
' ***CHANGE THIS LINE TO DEBUG...

If DEBUG_HOOK Then
    WScript.Echo "Objmenu/HookScript: Warning! DEBUG_HOOK is True"
End If

Public WShell    ' As Object
Set WShell = WScript.CreateObject("WScript.Shell")
If WShell Is Nothing Then
        WScript.Echo "Objmenu/HookScript: fatal error (unable to create WScript.Shell)"
        WScript.Quit 100
End If

Public oFS       ' As Scripting.FileSystemObject
Set oFS = WScript.CreateObject("Scripting.FileSystemObject")
If oFS Is Nothing Then
        ' vbOkOnly + vbCritical
        WShell.Popup "Unable to create Scripting.FileSystemObject", , "Objmenu/HookScript", 0 + 16
        WScript.Quit 101
End If

Call Main
```

Fig 1


Blaise 4.5
File Edit Project Run Database Tools Add-Ins Window Help
\\Lsurvey1\davispr\SQA\blaise api\old2new\ver1tover2.man
1: 1
Insert

```
MANIPULATE
  Outfile.QID:=Infile.QID
  Outfile.QDatabag:=Infile.QDatabag
  Outfile.Qtmdtot:=Infile.Qtmdtot
  Outfile.QStart:=Infile.QStart
  Outfile.QInter.QHealth:=Infile.QInter.QHealth
  Outfile.QInter.QEntdate:=Infile.QInter.QEntdate
  Outfile.QInter.QWhback:=Infile.QInter.QWhback
  Outfile.QInter.QLjbent:=Infile.QInter.QLjbent
  Outfile.QInter.QPaint:=Infile.QInter.QPaint
  Outfile.QInter.QNDExp:=Infile.QInter.QNDExp
  Outfile.QInter.QWhforw:=Infile.QInter.QWhforw
  Outfile.QInter.QLJBwl:=Infile.QInter.QLJBwl
  Outfile.QInter.QSearch:=Infile.QInter.QSearch
  Outfile.QInter.QBarrier:=Infile.QInter.QBarrier
  Outfile.QInter.QSComp:=Infile.QInter.QSComp
  Outfile.QInter.QChildCr.Introch:=Infile.QInter.QChildcr.Introch
  Outfile.QInter.QChildCr.CCpast:=Infile.QInter.QChildcr.CCpast
  Outfile.QInter.QChildCr.CCsch:=Infile.QInter.QChildcr.CCsch
  for i:= 1 to 16 do
     Outfile.QInter.QChildCr.Clcare[i]:=Infile.QInter.QChildcr.ccare1[i]
  enddo
  Outfile.QInter.QChildCr.ccarp1:=Infile.QInter.QChildcr.ccarp1
  Outfile.QInter.QChildCr.ccara1:=Infile.QInter.QChildcr.ccara1
  Outfile.QInter.QChildCr.ccbrak1:=Infile.QInter.QChildcr.ccbrak1
  for i:= 1 to 16 do
     Outfile.QInter.QChildCr.c2care[i]:=Infile.QInter.QChildcr.ccare2[i]
  enddo
  Outfile.QInter.QChildCr.ccarp2:=Infile.QInter.QChildcr.ccarp2
  Outfile.QInter.QChildCr.ccara2:=Infile.QInter.QChildcr.ccara2
  Outfile.Qinter.QChildCr.ccbrak2:=Infile.QInter.QChildcr.ccbrak2
  for i:= 1 to 17 do
```

Fig 2.

COM very briefly

COM stands for Component Object Model, or more precisely Microsoft® OLE Component Object Model. This technology has been part of the foundations of Windows application development for eight or so years. With the forthcoming .NET (dot-net) Framework, COM is being replaced by new and very different technology. Microsoft® are committed, however, to supporting COM for the foreseeable future such is the industry's reliance on what has been, and will remain, a fundamental technology in Windows development.

COM can be thought of as a 'Black Box' of functionality. A compiled program whose functionality can be accessed by other programs could be a COM Component. The internal implementation of the Component is of no concern. In order to build the driver's cockpit of a car the engineer does not need to know how to build an engine or how the front wheels steer, all she needs is to know how to connect the steering wheel and throttle to the relevant areas of the machine. COM displays a similar distinction between functionality that is hidden from the user - and Interface - exposed to the user. The Blaise API is presented to users via a COM interface. Any COM aware language can, therefore, bind to the component and make use of its functionality (see appendix A)

How might the Blaise API provide a better solution to data compatibility problems

The usual way in which a Blaise Datamodel is tested for compatibility with a database or another datamodel is basically to try it. If it works…then, well it works. If it doesn't then it's not compatible. Often this is a manual test: one which is neglected from time to time and has on occasion caused very time-costly problems. This is not ideal. The Blaise API Component provides us with the possibility of a much better solution, one which can be automated if need be and run behind the scenes without the involvement of Manipula or Cameleon.

What a COM solution looks like

One solution is to build another component. If we can wrap up the relevant functionality of the Blaise API together with the logic that suits our business environment then we have our solution. So at the core is BLAPI4A.DLL (the Blaise API). Sitting on top is our component developed in MS Visual Basic® 6.0, which currently has many different names - lets call it BLWRAP.DLL. On top of that is any client wishing to make use of it (remember we are still in COM-land so 'any client' really can be 'any'). Other major considerations are that the component is lightweight, simple and very easy to use.

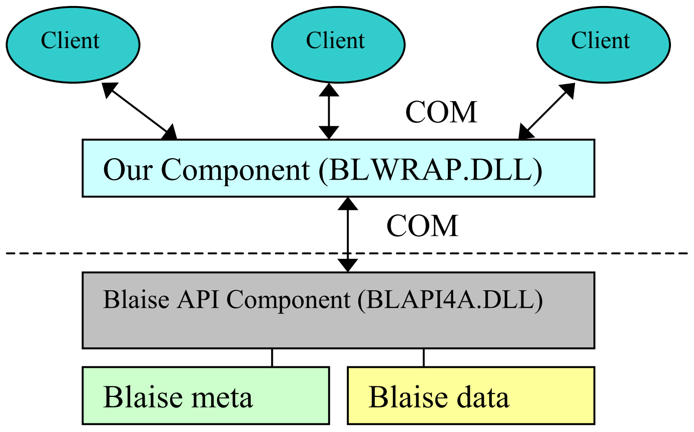

Client
Client
Client
COM
Our Component (BLWRAP.DLL)
COM
Blaise API Component (BLAPI4A.DLL)
Blaise meta
Blaise data

Fig 3.

Figure 3. shows the simple relationship between Blaise and our client application(s).

So the core of our solution has been identified as BLWRAP.DLL. This is what client applications will have to bind to if they wish to make use of the proposed functionality above. How is this functionality to be implemented in our component?

The COM Interface for our component

Simple, lightweight and very easy to use: these are all major goals in the design of our component. If it were over complicated then there would be no point using it - you may as well go straight to the Blaise API. For the purpose of this paper we will illustrate only the functionality concerning compatibility issues. As discussed earlier we need to have a mechanism through which two datamodels and data can be assessed for compatibility. In a simple usage the client may only need to know 'are they compatible?' A more demanding client may need to know 'Why not?' if the answer is 'No'.

A suitable interface definition could be a method as below:

```
     Compare(data1,data2) As Diffs
```

data1 and data2 are the datamodels to be compared, Diffs is a return collection of Difference objects. The Difference class describes a difference between data1 and data2 that results in incompatibility. This method can be used as follows;

```
'Constants to point to the two datamodels we are interested in.

Private Const DATA_MODEL1 As String= "C:\\WORKFILE\\DATA\\ONE0009A"
Private Const DATA_MODEL2 As String= "C:\\WORKFILE\\DATA\\ONE0009B"

     Sub Main()

          Dim cmpData As New BlaiseCompatibilty
          Dim cmpDiff As Difference
          Dim cmpDiffs As Diffs

          Set cmpDiffs = cmpData.Compare(DATA_MODEL1,DATA_MODEL2)

          If cmpDiffs.Count = 0 Then
               ' No differences - datamodels are compatible
          Else
               ' There are differences - datamodels are
               ' incompatible
               For Each cmpDiff In cmpDiffs
                    ' Iterate through the differences and process
                    ' the data as required.
               Next
          End If

     End Sub
```

Fig 4.

The ability to write small routines similar to Figure 4. is a great advantage. Almost every system we run will benefit from being able to dynamically assess compatibility between two datamodels - especially when they are seemingly the same version and previously would not have been checked. No in-depth benchmarking has currently been undertaken but on one of our medium length instruments a similar routine to the one above completes in a couple of seconds, quick enough not to be an issue.

Functionality of the component

How does our component use the Blaise API ? The reliability of our Compare method really depends upon one thing - what makes a datamodel incompatible with another? This paper started by highlighting some of the changes you can make to a datamodel which will render it incompatible with its data; there are doubtless more. This is an obvious area for further development. Once all the rules can be encapsulated within our component, it should be totally reliable.

Difference Objects are returned when the rules for compatibility are broken. The Difference Class discussed above is the member that describes the data specific to an incompatible difference between two datamodels. Internally, however, a Difference Object is created for every detail of a datamodel that may distinguish it from another.

Below is an overview of what happens when 'Compare' is called;

Using the Blaise API Component:

1. Open a database with DATA_MODEL1
2. Create Difference Objects for datamodel level attributes according to the rules.
3. Iterate through every field creating a Difference Object for each, storing its attributes according to the rules.
4. Close the database
5. Repeat 1-4 for DATA_MODEL2
6. Compare the two collections of Differences.
7. The method returns a collection of those which are not identical.

This is the basic mechanism behind the method. It does, however, perform collaborations with other areas of both our component and the Blaise API. It will, for instance, attempt conversion if required when incompatibilities are identified.

This section on functionality may seem to hold no substance - it really is that simple. Previously our Compare method could have been accomplished with various Cameleon and Manipula scripts. Without the messy parsing of the outputs, having the Compare method executing as part of another process and without user-input was impossible.

Summary

As this paper has shown the Blaise API Component offers us the opportunity to achieve real integration of Blaise functionality into our production systems. Our component is just one way to harness the core Blaise engine, it works for us and will no doubt continue to grow in functionality to provide an increasingly seamless integration of Blaise into our computing environment.

What's Next ?
Our agenda includes evaluating functionality for Active Server Pages, building a drag and drop datamodel generator for data conversion and a Blaise Custom Task Component for Microsoft® Data Transformation Services. There's a lot to do, and none-of it with Manipula or Cameleon.

Appendix A

Interoperability between languages is, chiefly, what COM is for. As previously mentioned; Microsoft® will not abandon the technology for a very long time. To highlight this fact the following is a console application which returns the number of records in a Blaise database. This simple application is written in C#. (Pronounced 'C Sharp' a core, brand new development language of Microsoft's impending .NET Framework)

```
using System;
using BlAPI4A;
namespace BlaiseApps
{
    class DataAttributes
    {
        static int Main(string[] args)
        {

            if (args.Length==0)
            {
             Console.WriteLine("Please enter a valid path and for a
                                          Blaise Database.");
             return 1;
            }

            DataAttributes MyClass = new DataAttributes();

            string error;

            double RetVal = MyClass.Records(args[0],out error);

            if (error.Length ==0)
            {
                Console.WriteLine ("{0} has {1} records.",args[0] ,RetVal);
                return 0;
            }
            else
                Console.WriteLine ("Blaise ERROR :" + error);
                return -1;
        }

        public virtual double Records(string FileName, out string error)
        {
            DatabaseManager MyManager = new DatabaseManager ();
            try
            {
                IBlaiseDatabase MyDataBase = MyManager.OpenDatabase(@FileName);
                MyDataBase.Connected = true;

                error = "";
                return MyDataBase.RecordCount;
            }
            catch(System.Runtime.InteropServices.COMException e)
            {
                error = e.Message ;
                return -1 ;
            }

        }
    }
}
```