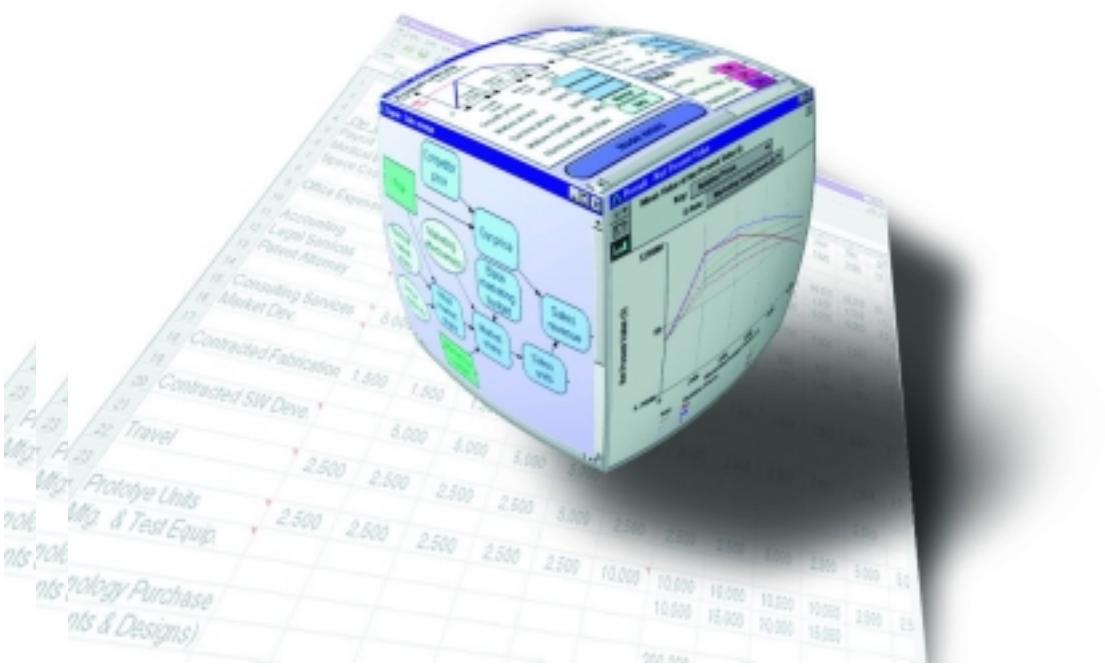
Analytica® Decision Engine for Windows



Tutorial

Release 3.1

November, 2004

Lumina Decision Systems, Inc.
26010 Highland Way • Los Gatos, CA 95033
(650) 212-1212 • www.lumina.com • support@lumina.com

Copyright notice

Lumina Decision Systems, Inc.

26010 Highland Way, Los Gatos, CA 95033

Tel: (650) 212-1212, Fax: (650) 240-2230,

E-mail: support@lumina.com

Internet: http://www.lumina.com

Acknowledgments

The Analytica Decision Engine Tutorial was written by Hugh Silin and modified for version 3.1 by Lonnie Chrisman.

Contents

Introduction

This tutorial demonstrates how to use the *Analytica® Decision Engine* (ADE) from within a Visual Basic program. It is assumed that you have properly installed both ADE (see the Installation instructions starting on page 3 of the Analytica Decision Engine for Windows Developer's Guide), and Analytica for Windows 1.2 or higher before reading this tutorial, and that your system meets all of the requirements described for those two products.

What is the Analytica Decision Engine?

ADE is a powerful tool that allows you to create, read, check, parse, evaluate, modify, and save Analytica models from within your own application programs. Although you can use ADE to prototype and refine your models, Analytica is much easier to use for this purpose (see the Analytica tutorial for information about creating and refining Analytica models). Following model refinement, you can use ADE to build an interface to your model from your program.

To provide the widest range of inter-application compatibility, ADE is provided as both an ActiveX in-process automation server (adew.dll) and an ActiveX local automation server (ade.exe). The classes, methods, and properties exposed by these servers are accessible from any OLE client compliant development environment (Visual Basic), any application with Visual Basic for Applications (VBA) support, including the Microsoft Office suite of applications, web pages using VB Script, JavaScript, or Microsoft Active Server Page (ASP) technology, and any C/C++ program.

Server objects allow you to read, check, parse, evaluate, modify, and save Analytica models from within your applications. For example, you can use Visual Basic to create graphical user interfaces (GUIs) on 32-bit Microsoft Windows platforms for your Analytica models tailored to specific applications and specific classes of end-users.

Using the Analytica Decision Engine Server

ADE exposes four OLE classes: `CAEngine`, `CAObject`, `CATable`, and `CAIndex` ('CA' stands for `Class Analytica`). You use these classes to interact with your model through ADE. The `CAEngine` class contains methods and properties that allow you to open and close existing models, create new models, and access Analytica objects contained in your model. The `CAObject` class contains methods and properties that allow you to set and obtain information about the Analytica objects that you obtain from the `CAEngine` class. Examples of Analytica objects are variables, modules, libraries, constants, and functions. The `CATable` and `CAIndex` classes are used to examine and modify definition attributes of Analytica index objects, and objects whose results are multi-dimensional.

Analytica Decision Engine Server Class Architecture

A conceptual model of ADE is presented in Figure 1 below. Your application makes calls to the functions exposed by the four interfaces of ADE. Those functions then return information to your application.

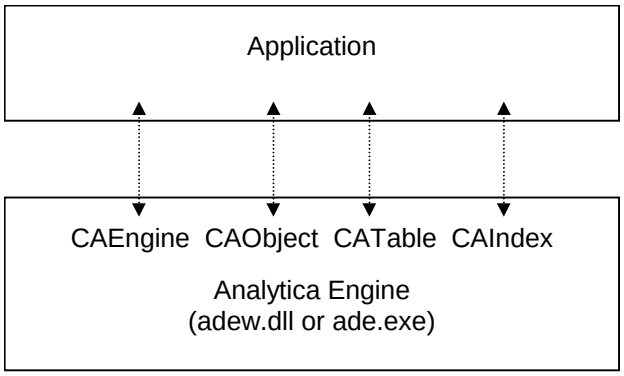


Figure 1. The Analytica Decision Engine Architecture

Your First ADE Application

Before we go further, let us write a simple ADE application from scratch, just to be sure that everything is set up correctly. We will assume you have Visual Basic.NET installed.

1. Bring up Visual Studio.NET

2. Select New Project, Project Type "Visual Basic Projects", and template "Console Application". Select a project name, e.g., "FirstADEtry" and an appropriate location.

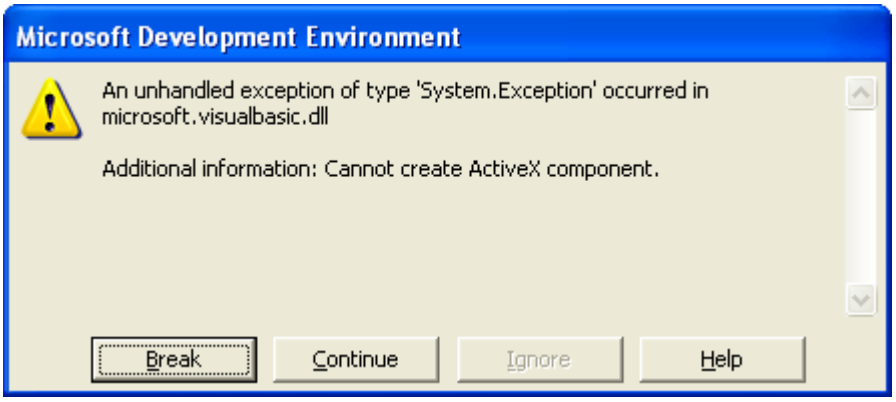
3. Add to the Module1 class as follows:

```
Module Module1
    Public m_ade As Object
    Sub Main()
        m_ade = CreateObject("ADE3.CAEngine")
        Dim filename, modelname As String
        filename = "C:\Program
Files\Lumina\Analytica 3.1\Tutorial Models\Car
Cost.ana"
        modelname = m_ade.OpenModel(filename)

        If modelname = "" Then
            Console.Write(filename & " not found")
        Else
            Console.Write("Congratulations on
opening " & modelname)
        End If
    End Sub

End Module
```

Run the program. If your program prints “Congratulations on opening Carcosts”, then you have just successfully written your first ADE program. If you received the following error message:



then ADE is not properly installed. Please consult the Analytica Decision Engine for Windows Developer's Guide for information about installing ADE before reading further.

In your first program, you successfully created a CAEngine automation object called m_ade (using CreateObject), opened an Analytica model (using the OpenModel method of CAEngine), and displayed the name of the model (the return value of OpenModel). We will go into the details of these functions, and many more functions in the next section.

What's next?

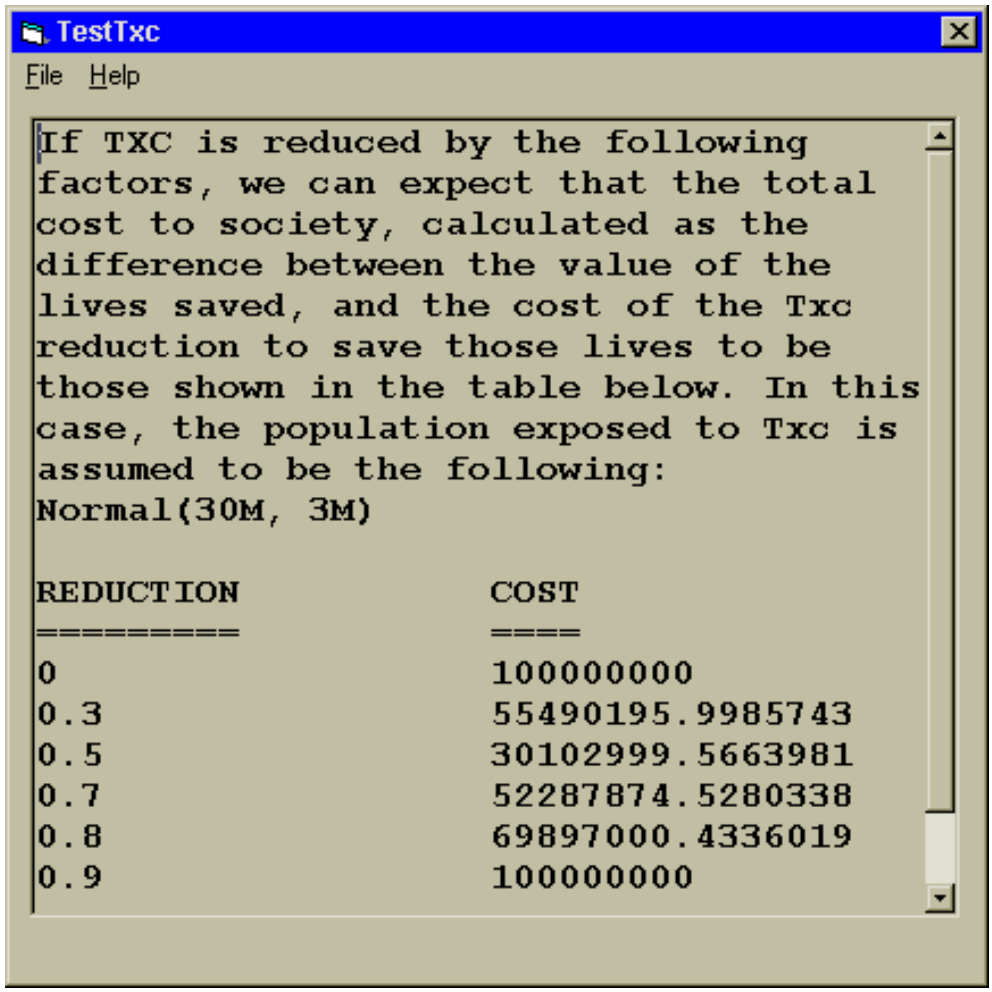
Because of the many things that you can do with ADE, this tutorial will not attempt to explain all of the features of ADE. Instead, it will give you all of the necessary background to explore the more advanced features of ADE on your own. To understand all of the features of ADE, please read the Analytica Decision Engine for Window's Developers Guide.

From this point, this tutorial uses the example model called Txc.ana to demonstrate how to use a lot of the features of ADE. Txc.ana can be found under the Risk Analysis folder under the Example Models folder of your Analytica product. If you cannot find Txc.ana, or if you opted not to install the examples when you originally installed Analytica, a copy of Txc.ana is included in the Examples\Tutorial folder under the directory where you installed ADE.

The Txc model demonstrates risk/benefit analysis, in this case regarding the benefits of reducing the emissions of fictitious air pollutant "TXC". Please load the Txc model into Analytica in order to gain an understanding of how the Txc model works.

The example Visual Basic program called TestTxc under your Ade Examples\Tutorial.NET folder demonstrates many aspects of ADE. (The Tutorial.NET example is for use with Visual Basic.NET, while the Tutorial.VB6 folder is for use with Visual Basic versions 5 or 6). In particular, it shows you how to create an ADE automation object, open the Txc.ana model with this object, get the definition of the "Population Exposed" variable, evaluate the "Total Cost" variable, print out the result of the "Total cost" variable as a table by getting at the individual components of the table, change the definition of the "Population Exposed" variable, and then get the result of the "Total cost" variable, again, to see what effect the change of

definition for “Population Exposed” had on the “Total Cost” variable If things are set up properly, then TestTxc displays the following window:



This application displays the definition of the “Population Exposed” object (Normal (30M, 3M)), and the table associated with “Total Cost”, based on the definition of “Population Exposed”. You can change the definition of “Population Exposed” by selecting File|Change Population Exposed from the main menu and seeing the effect this has on the “Total Cost” table.

Understanding the Difference Between Titles and Identifers

Whenever an ADE function requires the name of a variable, you must pass it the name of the identifier of the variable, and not its title. This can be confusing since Analytica, by default, displays the titles of each variable when a model is loaded. Also, if you don't specify an identifier for a variable when you first create that variable from within Analytica, Analytica will automatically create an identifier for the variable based on it's title.

To view the identifiers in the Txc.ana model, load the Txc.ana model into Analytica. Then, press Control-Y. Note how the titles of all the variables change to display the identifiers of the variables. You can see that the identifier of the variable titled "Population Exposed" is "Pop_exp". It is important to pass "Pop_exp" to all ADE functions that require a name for this variable. If you pass "Population Exposed" instead of "Pop_exp", ADE will not be able to find the variable, and an error will be returned.

Creating an ADE Object from within Visual Basic

If you have not already done so, load the project called Examples\Tutorial\TestTxc.vbproj into Visual Basic.NET, and view the code for the file called TestTxc.vb. It looks as follows:

```
Public adeEngine As Object

Public Sub Main()
    Dim exeDirectory, theModel As Object
    Dim theModelString As String

    exeDirectory = VB6.GetPath
    theModel = exeDirectory & "\" & "Txc.ana"
    adeEngine = CreateObject("ADEW3.CAENGINE")
        ...
    theModelString = adeEngine.OpenModel(theModel)
            ...
    frmMain.DefInstance.Show()
End Sub
```

At the very top of the file, the automation object called `adeEngine` is declared. This object is our `CAEngine` object. Through this object, we can access all of the public functions exposed by `CAEngine` (see the Analytica Decision Engine for Windows Developers Guide for a listing of all the functionality exposed by `CAEngine`). To create our ADE automation object, the following is done:

```
adeEngine = CreateObject("ADEW3.CAENGINE")
```

Note: In VB 5 or 6, this must be written as:

```
set adeEngine =
CreateObject("ADEW3.CAENGINE")
```

This assigns our in-process ADE server object to our `adeEngine` object. If we want to use the local server version of our ADE server, we would, instead, make the following call:

```
adeEngine = CreateObject("ADE3.CAENGINE")
```

Note the difference between the strings passed to `CreateObject`. If “ADEW.CAENGINE” were passed into `CreateObject`, the in-process version of ADE would be used. If “ADE.CAENGINE” were passed into `CreateObject`, then the local server version of ADE would be used. For a brief

description of the benefits and deficiencies of using a local server vs. an in-process server, as well as which automation server you should use under different scenarios, please consult the Analytica Decision Engine for Windows Developers Guide, as well as other books related to OLE automation servers.

COM vs. Automation Interface

In the above examples, an ActiveX Automation interface was used to call ADE. In an automation interface, the object (CAEngine in this case) is declared as Object and member functions are resolved at run-time. In other words, when you compile your program, the compiler does not know whether adeEngine has a method named OpenModel or how to call it. At run-time, when the call to adeEngine.OpenModel is encountered, the method is found and called.

It is also possible from Visual Basic to use a COM interface. The COM and Automation interfaces are different underlying technologies for calling the methods of ActiveX objects. When a COM interface is used, the compiler resolves each member function and can detect several obvious errors at compile time. In addition, the Visual Studio can provide a list of methods and parameter types as tool tips as you program. COM calls are slightly more efficient than Automation calls, but the speed difference is typically not a significant factor in applications of ADE. To use the COM interface, you must first add a reference to the ADE engine to your project in Visual Studio by selecting "Project" / "Add References...", clicking on the "COM" tab, and finding "Analytica Decision Engine in-process Server", TypeLib version 3.1. Once this is selected, change the declaration of the ADE object(s) to an explicit type, such as:

```
Public m_ade As ADEW3.CAEngine
```

When you create the m_ade object, you can now use:

```
m_ade = New ADEW3.CAEngine
```

Once the object is created, the syntax for calling methods is identical whether you are using Automation or COM.

In environments other than Visual Basic, you may find that only Automation or COM calls are supported, but not both. For example, in VBScript or JScript, only Automation is used. In C++, the COM interface is generally much more convenient, while Automation requires a rather tedious use of the IDispatch interface.

Opening a Model with ADE

We will now open the Txc.ana model, and show the main window of our application. This is done with the following call:

```
theModelString = adeEngine.OpenModel(theModel)
frmMain.DefInstance.Show
```

Note: In VB 5 or 6, the second line is just: frmMain.Show

The OpenModel function of CAEngine will open the model. If successful, then the variable theModelString contains the name of the model. Otherwise, it contains an empty string. Although we haven't done so in this example for the sake of brevity, you should check to see that the string returned from OpenModel is not an empty string. If it is, then an error in opening your model has occurred. The error is obtained by using the ErrorCode and ErrorText properties of CAEngine (adeEngine.ErrorCode and adeEngine.ErrorText). We will show how to use these two properties later on in this tutorial. For a listing of all the error codes, please consult the Analytica Decision Engine for Windows Developer's Guide.

Retrieving Objects from the Analytica Model

Now that we have opened our model, the next step is to retrieve objects (e.g., Variables, Modules, Functions, etc.) from our model in order to evaluate and retrieve their attributes (definition, title, class, etc.). Our example model (Txc.ana) manipulates the Pop_exp and Cost objects. In particular, it modifies Pop_exp to see how this affects the Cost object.

The PrintAttributes function in the file frmMain.frm of our TxcTest.vbproj (TxcText.vp) project demonstrates how to do this. This function is called initially by the Form_Load function of frmMain.frm when the application first starts in order to display the current value of our Cost table. It is also called whenever we wish to print out the current result of our Cost table. The function looks as follows:

```
Public Sub PrintAttributes(ByRef inputIdentifier As
        String, ByRef outputIdentifier As String)
    Dim inputObject, outputObject As Object
    Dim resultTable As Object
    Dim definitionAttrInput As String

    inputObject =
        adeEngine.GetObjectByName(inputIdentifier)
    outputObject =
        adeEngine.GetObjectByName(outputIdentifier)
    definitionAttrInput
     inputObject.GetAttribute("definition")
    resultTable = outputObject.resultTable
    Call PrintResultTable(resultTable,
        inputIdentifier, definitionAttrInput,
        outputIdentifier)
End Sub
```

This function starts by getting at the objects associated with Pop_exp and Cost (the inputIdentifier parameter is passed in as "Pop_exp" and the outputIdentifier is passed in as "Cost"). These are fetched using the GetObjectByName function of CAEngine by the following lines of code:

```
inputObject =
      adeEngine.GetObjectByName(inputIdentifier)
outputObject =
      adeEngine.GetObjectByName(outputIdentifier)
```

Note: In VB 5 or 6, these lines must be preceded by `set`.

If GetObjectByName succeeds, it returns an object of type CAObject. You then use the functions of CAObject. Please consult the Analytica Decision Engine for Windows Developer's Guide for a listing all CAObject functions. If GetObjectByName fails, then the return value will be `Nothing`. Although it is not done in our sample (for the sake of brevity), code should check to be sure that the object being returned from GetObjectByName is valid. If it is not valid, use the ErrorCode and ErrorText properties of CAEngine to get more information about the error. So, you might want to write the following code:

```
Set inputObject =
           adeEngine.GetObjectByName(inputIdentifier)
If inputObject Is Nothing Then
     Dim ErrorString As String

     ErrorString = "The following error from "
     ErrorString = ErrorString & "GetObjectByName
           occurred: " & vbCrLf
     ErrorString = adeEngine.errorCode & ":"
     ErrorString = adeEngine.errorText
     MsgBox ErrorString
Else
     'inputObject valid
End If
```

Getting the Attributes of Objects in the Analytica Model

Once we get at the objects of interest in our model (which are returned as CAObject interfaces), we will query some of the attributes of those objects. For example, to get the definition of the “Pop_exp” object, use the following:

```
definitionAttrInput =
            inputObject.GetAttribute("definition")
```

In the Txc.ana model, the definition of Pop_exp is “Normal (30M, 3M)” which is returned in definitionAttrInput. There may be numerous attributes associated with an object. For example, you can also retrieve the title, description, and class of an object (as well as many other attributes). For a complete list of attributes associated with an object, please consult the Analytica User’s Guide that came with Analytica.

Evaluating Objects and Retrieving Results

To evaluate an object, use the Result and ResultTable functions of CAObject. Result is generally used to retrieve the results of scalar objects, while ResultTable is used to retrieve the results of multi-dimensional objects. By default, Result and ResultTable return the mid value of the result (consult your Analytica User's Guide for an explanation of the different result types that can be returned when evaluating an object). To return the result as a probabilistic value, or as a mean, set the ResultType property of CAObject. Please see the Analytica Decision Engine for Windows Developer's Guide for more information about ResultType. We retrieve the result of our Cost object (outputObject) in the following way:

```
resultTable = outputObject.ResultTable
```

Note: In VB 5 or 6, use: `Set resultTable = ...`

This function returns a CATable object that allows us to access (and set) individual elements in a table. Below, it is shown how to use the function GetDataByElements, which is one function that can be used to access individual elements in a table. Here, use the function ResultTable because we know that our result is multi-dimensional. If ResultTable did not evaluate to a multi-dimensional result, then `Nothing` would be returned by ResultTable. This could mean that either an error had occurred or that the object in question represents a scalar value (in which case the Result function should be used).

Sometimes it is hard to tell whether a result will be scalar or multi-dimensional. For example, an object in your model may evaluate to a scalar under some circumstances, and a multi-dimensional result in other circumstances. In this case, call ResultTable first. If ResultTable succeeds, then you are dealing with a multi-dimensional value. If ResultTable returns Nothing, then call Result. If Result succeeds, then you are dealing with a scalar. If Result fails, then an error has occurred. Call the ErrorCode and ErrorText functions to understand why an error occurred, and take appropriate action.

For example, you might write code that looks as follows:

```
Dim result As Variant
Dim resultTable As Object 'our CATable
resultTable = outputObject.ResultTable
If resultTable Is Nothing Then
     'Check to see if it is a scalar result
     result = outputObject.Result
     If adeEngine.ErrorCode <> 0 Then
          'an error occurred.  Take action
     Else
          'continue with result
     End If
Else
     ‘continue with resultTable
End If
```

Note that in the above code snippet, ResultTable returns a CATable object. To access the individual elements of the table, use the functions exposed by the CATable object. The function Result, however, returns either a string or a number, and not an automation object. You can, therefore, use the return value of Result directly.

Although ResultTable is generally used with multi-dimensional objects, you can also use the Result function for multi-dimensional objects. This, however, is very rarely done because Result, in the case of a multi-dimensional result, returns a string representation of your table, which can be very hard to parse if you need to get at individual elements of the table.

Getting the Index Elements of a Table

All Analytica tables have indexes associated with them. If there is one index associated with a table, then the table is one-dimensional, if there are two indexes associated with a table, then the table is two dimensional, etc. (you can use the NumDims function of CATable to determine the dimensionality of your table). To get at the individual indexes of a table, make use of the functions IndexNames and GetIndexObject of CATable. The function PrintResultTable in frmMain.frm demonstrates the use of these two functions (as well as other concepts that we present below). PrintResultTable is called from PrintAttributes, and does the actual work of printing the table that shows up in our TestTxc application. Note that only the parts of this function related to ADE are shown for brevity.

```
Public Sub PrintResultTable(ByRef resultTable As
        Object, ByRef inputIdentifier As String,
        ByRef definitionAttrInput As String, ByRef
        outputIdentifier As String)
    Dim theIndexName, theTableName As String
    Dim theIndexElement As String
    Dim theTableElement
    Dim theIndexObj As Object
    Dim numEls As Integer
    Dim spaces, i As Integer
    Dim lenStr As Short
    Dim OutputStr As Short
    Dim spaceString, underlineString As String
                .
                .
    theIndexName = resultTable.IndexNames(1)
    theTableName = resultTable.Name

    theIndexObj =
        resultTable.GetIndexObject(theIndexName)
    numEls = theIndexObj.IndexElements
                .
                .
    For i = 1 To numEls
        theIndexElement =
        theIndexObj.GetValueByNumber(i)
        theTableElement =
        resultTable.GetDataByElements(i)
                .
                .
    Next i
    InformationPane.Text = outputString
End Sub
```

The lines of PrintResultTable that get an index of a table are as follows:

```
theIndexName = resultTable.IndexNames(1)
theIndexObj =
resultTable.GetIndexObject(theIndexName)
```

The first index in the table is retrieved by using the IndexNames function of CATable (if this were more than a one-dimensional table, and we wanted the name of the second index, we would pass 2 into the IndexNames function rather than 1). We then pass the name of this index into the GetIndexObject function of CATable to retrieve a CAIndex object that represents our index. This automation object returns information about its corresponding index. If this function fails, then `Nothing` is returned. In this case, the ErrorCode and ErrorText functions of CAEngine should be used to gain more information about the error.

Retrieving Information from our CATable and CAIndex objects

PrintResultTable also demonstrates how to retrieve information from CATable and CAIndex objects. The code, which gets at the index elements and the table elements of our Cost table, is as follows:

```
numEls = theIndexObj.IndexElements
For i = 1 To numEls
    theIndexElement =
            theIndexObj.GetValueByNumber(i)
    theTableElement =
            resultTable.GetDataByElements(i)
        .
        .
Next i
```

The IndexElements property of CAIndex returns the number of elements in the index (the first index).

The GetValueByNumber function of CAIndex retrieves individual index elements of the index.

To retrieve the individual table elements of the Cost table object, resultTable, the GetDataByElements function of CATable is called (passing in the position of the table we wish to retrieve).

In our example, when we retrieve an individual element of our CATable object (resultTable), we take advantage of the fact that the table is one-dimensional. Therefore, we only need to pass GetDataByElements a single number representing the position in our table. If we were dealing with two or more dimensions, however, we would need to pass GetDataByElements an array specifying the coordinate of the element of our table to retrieve. So, if we want to retrieve the element at position 4,3 of a 2-dimensional table, we would write the following code:

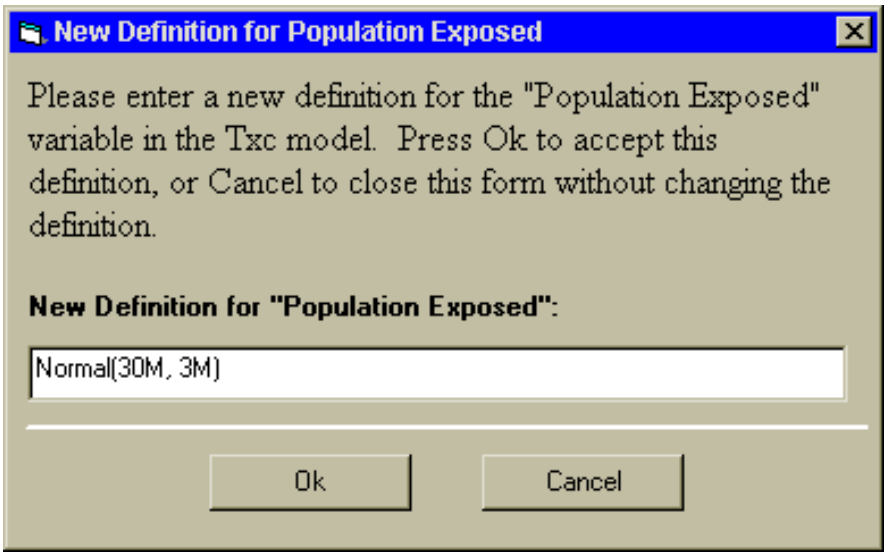
```
Dim W as Variant 'return element
Dim IndexPtrs(1 To 2) As Variant 'position in
            table
....
....
IndexPtrs(1) = 4
IndexPtrs(2) = 3
W = resultTable.GetDataByElements(IndexPtrs)
```

There are other ways to retrieve information from CATable and CAIndex objects. Please consult the Analytica Decision Engine Reference Guide for the functions exposed by the CATable and CAIndex objects.

Modifying Objects

A custom application typically obtains input from a user or other external source, and then transfers this data to input variables in their Analytica model before retrieving results from the model. Input data is usually transferred into ADE either by setting the definition of an input variable, or by using a definition table.

The TestTxc shows how to modify the definition of pop_exp, which is a model input that affects the Cost result variable. To set the definition in the example, select File|Change Population Exposed from the main menu. The following dialog appears:



In the “New Definition for “Population Exposed” field, you can enter a new definition for pop_exp, and press OK. At that point, the main window of our application will display the new result of Cost. The OkButton_Click function in ChangeDef.frm is called when the Ok button is pressed in the above dialog. It modifies the definition of pop_exp, and then calls the PrintAttributes function that prints the result of Cost.

The function looks as follows:

```
Private Sub OkButton_Click(ByVal eventSender As
        System.Object, ByVal eventArgs As
        System.EventArgs) Handles OkButton.Click
    Dim errorText As String
```

```
    Dim pop_exp_Object As Object
    Dim errorCode As Short
    Dim errorString As String
    Dim newDefinition As String

    newDefinition = PopExposedDef.Text
    pop_exp_Object =
        adeEngine.GetObjectByName("pop_exp")
    pop_exp_Object.SetAttribute("definition",
        newDefinition)
    errorCode = adeEngine.errorCode
    If errorCode <> 0 Then
        errorString = adeEngine.errorText
        errorText = "The following error occurred
        while processing your definition: " & vbCrLf
        & vbCrLf
        errorText = errorText & errorString
        MsgBox(errorText)
        PopExposedDef.Focus()
    Else
        Me.Close()

        frmMain.DefInstance.PrintAttributes("Pop_exp"
        , "Cost")

    End If
End Sub
```

This function grabs the new definition typed into the New Definition for Population Exposed field and sets it to the “pop_exp” object by using the SetAttribute function of CAObject object. It then calls PrintAttributes, which evaluates the Cost object, and prints the new table.

To set a new definition for the pop_exp variable, we get the CAObject for pop_exp, and set it’s definition to the definition typed in by the user. This is done with the following code:

```
pop_exp_Object =
        adeEngine.GetObjectByName("pop_exp")
 pop_exp_Object.SetAttribute "definition",
        newDefinition
```

Whenever you call SetAttribute, you should check the ErrorCode of the CAEngine automation object (adeEngine), in case the definition is illegal.

Try entering a new definition such as Uniform(25M,35M) and press Ok. When the definition of pop_exp is changed, the result for Cost gets recomputed by ADE when ResultTable is next called for the Cost variable (when the application window is repainted).

Conclusion

This tutorial touched on several aspects of the Analytica Decision Engine. It explained how to create the ADE server object, open a model with ADE, get at an individual object in a model, evaluate objects, access elements in a table, and modify objects in a model. It is important to note that this tutorial has barely scratched the surface of the things that can be done with ADE. However, we hope that you have gained enough information about the basics of ADE in order to explore the many advanced features of ADE on your own. It is strongly recommended that you read the Analytica Decision Engine for Windows Programmers Guide in order to gain a complete understanding of ADE, and what it can accomplish. Good luck!!