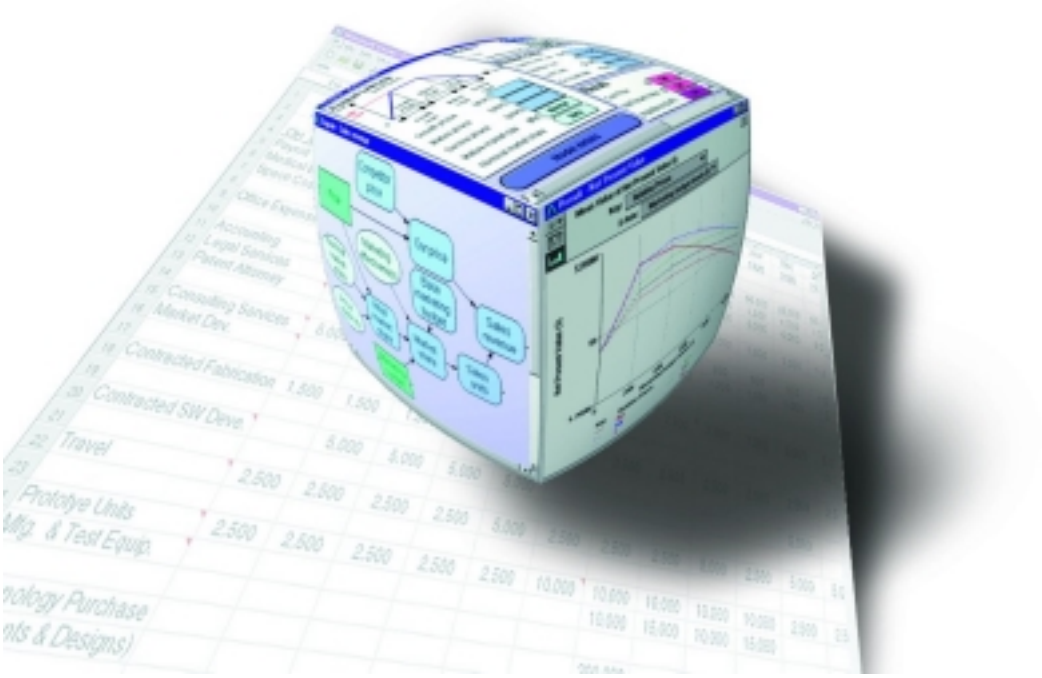
Analytica® Decision Engine for Windows



Developer's Guide

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 for Windows Developer's Guide was written by Richard Sonnenblick, Hugh Silin, and Lonnie Chrisman.

Contents

Introduction

Thank you for purchasing the *Analytica Decision Engine for Windows* (ADE) developer's kit. This document describes how to use ADE. If you are a Visual Basic or C++ programmer interested in accessing information in your Analytica models from within your applications, from within Microsoft applications such as Excel and Word, or from Microsoft ASP, this document will help you get started.

What is the Analytica Decision Engine?

The *Analytica Decision Engine* (ADE) is a powerful ActiveX component that helps you to programmatically interact with and create Analytica models. The engine can create, read, check, parse, evaluate, modify, and save Analytica models. Although you can use ADE to prototype and refine your models, we recommend that you use Analytica for this purpose (see the Analytica tutorial and reference guide for information about creating and refining Analytica models). Following model refinement, you can use ADE to build an interface to your model.

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, any C/C++ program, and many others. 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 or C/C++ 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.

How to use this document

Following this introduction, this document is divided into three sections:

1. Installation

This section explains the steps required to install the Analytica Decision Engine 3.1 on your Windows 98, ME, NT 4 (>SP 6), 2000, or XP computer.

2. Using the Analytica Decision Engine Server

This section provides a step-by-step guide to the functionality accessible through ADE. You should read this section to get better acquainted with the classes, and their methods and properties. By using the sample code fragments presented in this section in your code, you can begin accessing information in your models from your Visual Basic applications immediately.

3. Analytica Decision Engine Server Class Reference

This section provides reference materials on the four object classes in ADE and their properties and methods. Information that can be found in this chapter includes method syntax, data types, and property access information. Refer to the information in this section after you've read through the section, "Using the Analytica Decision Engine Server", and have specific questions about particular methods and properties.

Installation

System Requirements

Windows 98, ME, NT 4.0 (>SP6), 2000, or XP.

Ten Mb of hard drive space (additional space will be required for development of your applications).

Eight Mb of RAM (16 Mb recommended for Visual Basic Applications development)

In addition to the above requirements, you will need an OLE-automation enabled development environment such as Visual Basic.NET, or Visual Basic 5 or 6 Standard Edition (minimum), 32-bit version.

The files required for installation of ADE are shipped on a CD or downloaded over the network. The installation contains the ADE in-process automation server (adew.dll), the ADE local automation server (ade.exe), analytica.i and Analytic.ini (which are needed by adew.dll and ade.exe), and four example programs.

Installing the Analytica Decision Engine files

[Installing from the network]

1. Obtain an ADE 3.1 license code from Lumina. This will be supplied to you, usually through e-mail, when you purchase ADE. You must complete the installation within the specified number of days after the license code is issued to you, or it will become stale. A fresh code can be obtained by following the directions accompanying the license code.
2. Download the ADE setup executable. The location of the file will be provided to you when you receive your license code. Save the file to disk.
3. Run (e.g., double click on) the file just download to begin the ADE installer.
   Note: Before running the ADE installer, you must have the Windows System Installer (WSI) already installed on your system. This is guaranteed to already be on your system if you are using Windows XP, or if you have previously installed Analytica 3.1. If you have installed any recent Windows

software, it is also very likely to already be present. If it is not present, the easiest remedy is to install Analytica 3.1.

4. Follow the instructions. Read and agree to the license agreement, select a directory for the installation, and enter your license code when prompted.
   If the installer reports that your license code is stale, go to http://lumina.com/ana/refreshLicense and obtain a fresh code. After you obtain a fresh code, be sure to enter the license code within three days.

[Installing from CD]:

1. Insert the Analytica Decision Engine CD into your CD-ROM.
2. If the installer does not automatically start, run the setup.exe program on the CD-ROM.
3. During the setup, you will need to select a directory for installation, to read and agree to the licensing terms, and to enter the license code supplied to you by Lumina Decision Systems when you acquired ADE.

   If the installer reports that your license code is stale, go to http://lumina.com/ana/refreshLicense and obtain a fresh code. After you obtain a fresh code, be sure to enter the license code within three days.

After following the above steps, the following files should exist in the directory in which you installed ADE:

```
Adew.dll
Ade.exe
Analytic.ini
Analytic.i
MathFunctions.dll
ArrayFunctions.dll
FinancialFunctions.dll
ODBC4Analytica.dll
```

Three ADE manuals are installed into a subdirectory called docs. These are:

ADE Developer's Guide.pdf (this document)
ADE Tutorial.pdf
ADE Scripting Guide.pdf

Six example programs should also have been installed in that directory underneath the examples directory. They are as follows:

Tutorial.VB6 and Tutorial.NET- (separate versions for users of VB versions 5/6 and VB.NET) the program referred to by the Analytica Decision Engine Tutorial. It is recommended that you read the Analytica Decision Engine Tutorial completely before writing your own programs that depend on ADE.

Anatest- a program that tests the functionality of ADE for users of VB 5 and 6. You can run this program inside of the Visual Basic version 5 or 6 debugger to gain some insight into the proper syntax of some of ADE's calls. This program makes use of the vast majority of the functionality exposed by ADE.

AdeTest – a program that allows you to call or test the methods of ADE objects. You can run AdeTest.Exe (in the bin directory), or you can trace through the code in the Visual Basic.NET debugger to observe each method being called.

asp_exam – a program that shows how to access ADE through a Microsoft ASP program.

excel_exam – a program that shows how to access ADE from any application with Visual Basic for Applications (VBA) support, including the Microsoft Office suite of applications.

Entering a new license code

If you have previously installed ADE 3.1 and need to enter a new (different) license code, follow these steps:

1. Open a command prompt.

   On Windows 98 or ME, select Start → Run and type Command.exe.

   On Windows NT 4, 2000, or XP, select Start → Run and type Cmd.exe.

2. Change directory (using the cd command) to the directory where you installed ADE 3.1.

3. Type: ADE /RegServer

A dialog will appear prompting you enter your new license code.

Uninstalling ADE

To uninstall ADE, select “Add/Remove Programs” in the Windows Control Panel. Scroll to find ADE 3.1. Press the “Change/Remove” or “Add/Remove…” button, depending on your operating system, and select “Remove”.

The uninstall will only remove files that were placed on your system by the installer. If you have compiled some of the examples, there may be some files and directories, created during those exercises, which are not removed. To remove these as well, find the install directory (e.g., C:\Program Files\Lumina\ADE 3.1) and remove it (but only after the uninstall has been run).

Using the Analytica Decision Engine Server

ADE exposes four automation objects: `CAEngine`, `CAObject`, `CATable`, and `CAIndex` (‘`CA`’ stands for `Class Analytica`). The `CAEngine` class contains methods and properties that allow you to open and close existing models, create new models, create new Analytica objects, 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. The `CATable` and `CAIndex` classes are used to examine and modify definition attributes of Analytica index objects, and objects whose results are multi-dimensional. The following sections describe how to access these Analytica Server objects from Visual Basic.

Analytica Decision Engine Server Class Architecture

In addition to the Program Interface, ADE has a fully functional command interface, known as the typescript. The typescript language is described in the *Analytica Scripting Guide,* and allows access to all of ADE’s functionality. The API provides a more convenient, object-oriented, set of functions for communication with the engine from Visual Basic and C++ applications. A calling program can use the API functions, or it can pass typescript commands directly to the typescript interface.

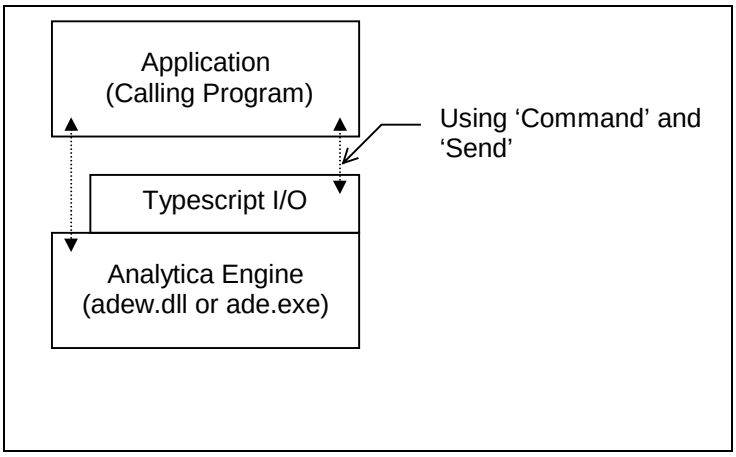
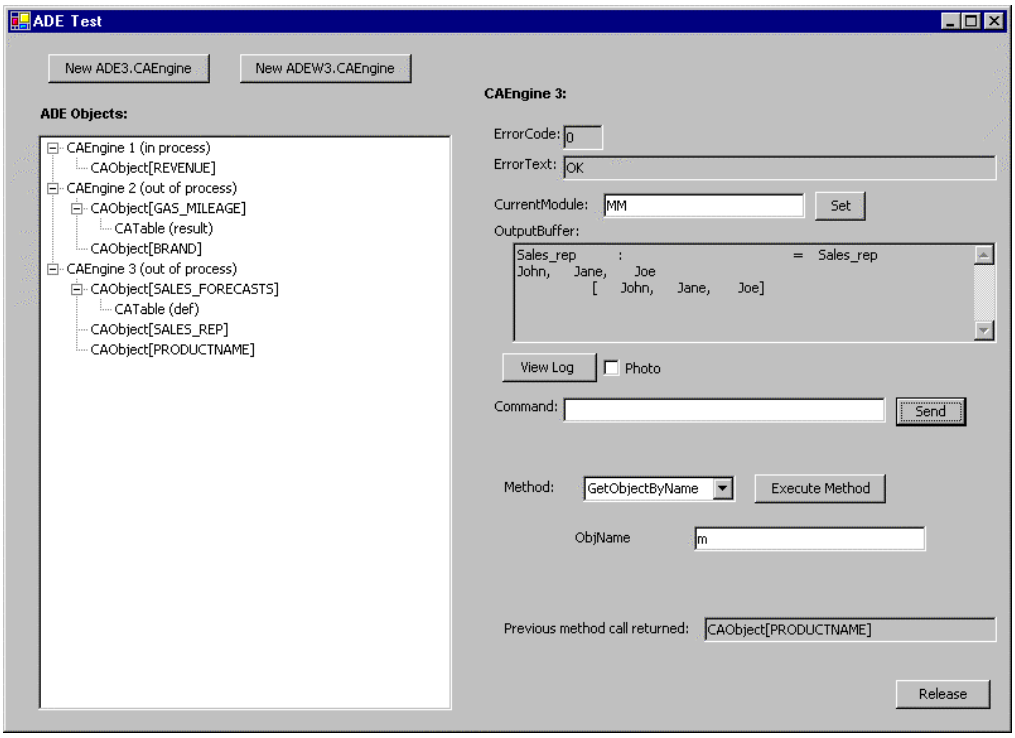


Figure 1. The Analytica Decision Engine Architecture

The AdeTest Program

ADE 3.1 ships with a sample program called `AdeTest.exe`. The executable can be found in the Examples/AdeTestbin directory. You can use `AdeTest` to exercise the functionality of either the in-process (adew.dll) or the local process (ade.exe) versions of ADE 3.1. Using `AdeTest`, you can send script commands to the engine, create ADE objects, set or call virtually any of the properties and methods of the ADE objects. If you have Visual Basic.NET installed, you can step through the code in the Visual Studio Debugger to observe the methods being called.

The diagram shows a screenshot of the AdeTest program. The left-hand pane shows a list of ADE objects that the program is currently holding. The right side shows details of one of those objects. In the screenshot, there are three `CAEngine` instances, each with a different model open. The first `CAEngine` is an in-process (ADEW.DLL) instance, while the second two are out-of-process local servers (ADE.EXE) instances. The two buttons above the left pane can be used to create additional `CAEngine` instances, while the Release button at the lower-right corner of the right-hand panel releases an instance. The right-hand panel shows information about the third `CAEngine` instance. The current values for the `CAEngine` properties `ErrorCode`, `ErrorText`, `CurrentModule`, `OutputBuffer` and `Photo` are displayed. You can execute a typescript command by typing the command into the text box area and pressing the “Send” button. Or you can execute any of the method of `CAEngine` by selecting the method in the drop-down Method box, filling in the parameters and pressing the “Execute Method” button.



If you click on an object in the left-hand pane, the properties for that object will be displayed on the right-hand side and properties can be set or its methods called. Thus, you can simulate a series of steps your program might execute through the graphical interface.

When a method returns an object, for example, as with `CAEngine::GetObjectByName`, the object returned is added to the tree on the left as a child of the object that created it. After executing a method from a class other than `CAEngine`, it is a good idea to glance at the corresponding `CAEngine's` panel to check the `ErrorCode`, `ErrorText`, and `OutputBuffer` properties.

The Photo checkbox in the Analytica window is mirrored by the Photo property of the `CAEngine` class. By default the `Photo` property is `False`, so typescript communications between the client and ADE are not copied to the Analytica Log Window. Setting the `Photo` property to `True` will copy all subsequent typescript communications between the client and ADE. In Visual Basic, this would be done as follows:

```
Ana.Photo=True
Ana.Photo=False
```

Turning the `Photo` property on significantly slows down communication with ADE.

Sample Application in Excel's Visual Basic

Another example program called `excel_exam` is also included in the ADE package. The program, `Analytica.xls`, in the excel_exam directory can be loaded into Microsoft Excel and executed as a macro. This program demonstrates the use of Visual Basic for Applications in Excel for ADE communications. This sample makes use of the local server version of ADE.

When using Microsoft's ASP, we recommend that you use the local server. By using the local server (ade.exe), you can ensure that each web application, or even each session, uses a different version of ade.exe. Currently, there is a limitation in ADE that prevents creation of two or more in-process server objects at the same time. Therefore, if you expect to have more than one session of ADE active at one time (as is almost always the case in web-based applications, always use the local server of ADE. An example program called `asp_exam` shows how to use the local server of ADE with a Microsoft ASP application.

To create a new Visual Basic project which uses the in-process server of ADE:

Start Visual Basic.

Create a new project [or document], using the File/New command.

Next, add a link to the Analytica C++ Engine Server 3.1. Use the References… option of the Project pull down menu [or the Project/References item]. Put a check next to the 'Analytica C++ Engine Server 3.1' selection. Note that this is different from the 'Analytica C++ Engine Local Server 3.1' option. The 'Analytica C++ Engine Local Server 3.1' option cannot currently be used within Visual Basic.

The four Analytica classes `CAEngine`, `CAObject`, `CATable`, and `CAIndex` are now accessible from your project. You can use all of the methods and access the class properties described in this reference document in your Visual Basic application. Only the `CAEngine` class will allow object creation from your program; the `CAEngine` class, in turn, creates and manages objects of the three

other classes. The following section describes how to create an object of type `CAEngine`.

Initializing ADE

Before an Analytica model can be opened or created, you must initialize the Analytica Decision Engine by creating an object of type `CAEngine`.

From a Visual Basic.NET program that uses the in-process version of the ADE, initialize ADE using a `New` statement, as demonstrated below.

```
Dim Ana as ADEW3.CAEngine
Ana = New ADEW3.CAEngine
```

From a Visual Basic version 5 or 6 program, use the syntax:

```
Dim Ana as CAEngine
Set Ana = New CAEngine
```

Note that for the above to work, you must have added the Analytica Decision Engine 3.1 in-process server to the project references.

From an MSOffice application, initialize the in-process version of ADE in the following way. Note the use of `Object` instead of `CAEngine`. All of the ADE automation components (`CAEngine, CAObject, CATable`, and `CAIndex`) must be referred to as Objects when using the `CreateObject` function.

```
Dim Ana as Object
Set Ana = CreateObject("ADEW3.CAENGINE")
```

When using the local server version of ADE, initialize the Analytica Decision Engine in the following way (note the use of `ADE3.CAENGINE` rather than `ADEW3.CAENGINE`):

```
Dim Ana as Object
Set Ana = CreateObject ("ADE3.CAENGINE")
```

This code snippet results in a new object `Ana`, an instance of class `CAEngine` that initializes ADE (from an ASP program, of course, you would just do `Dim Ana` , not `Dim Ana As Object`).

When using the in-process version of ADE, only one CAEngine object can exist at a time. However, multiple versions of the CAEngine object can exist at one time when using the local server.

After initializing a CAEngine object you can:

- Use the Command property and the Send method to send typescript commands to ADE and receive resulting output from the Engine. Typescript commands are described in the *Analytica Scripting Guide* and can be used to access all the functionality of ADE.
- Use the methods and properties of classes CAEngine, CAObject, CATable, and CAIndex to communicate with ADE.

After the calling application has closed the current model, it may destroy the Ana object using the CAEngine class destructor:

```
Set Ana = Nothing
```

The above statement terminates this instance of ADE, freeing all memory associated with this instance of ADE.

ADE Typescript: Command Language Communication

The Command property and Send method of the CAEngine class allow you to use typescript commands, sent as ASCII strings to the Engine, and receive the resulting output as another ASCII string. You may want to use a typescript command instead of an API method if:

- You want to perform your own parsing on ADE output (e.g., on tabular data that are output from the Analytica Decision Engine as text strings of comma-delimited text).
- No appropriate API method exists.

You perform three steps to send a typescript command to ADE:

1. Assign a text string containing the command to the Command property of your CAEngine object.
2. Use the Send method to send the command to the Engine. If the Send method returns True, then the command was processed without error by ADE.

3. Store the error code and error text (if the return code is nonzero). These two pieces of information are stored in the `CAEngine` properties `ErrorCode`, and `ErrorText`.
4. Get the output by calling the `OutputBuffer` function in the `CAEngine` class.

The steps are demonstrated in the following Visual Basic code segment:

```
Dim RetCode, SendCode as Boolean
Dim Result As String
Dim ErrCode as Integer
Dim ErrT as String

Ana.Command = "probvalue var1" 'any typescript command
SendCode = Ana.Send
If SendCode = False Then
      ErrCode = Ana.ErrorCode
      ErrT = Ana.ErrorText
Else
      Result = Ana.OutputBuffer
End If
```

Errors and Error Handling

The `CAEngine` properties `ErrorCode` and `ErrorText` should be queried after any operation with ADE whenever an error is possible. The possible error codes in `ErrorCode`, and the associated string in the `ErrorText` property is described below.

Error Code (ErrorCode)	**Meaning (**ErrorText**)**
0	"OK"
1	"Unimplemented"
2	"Warning"
3	"Lexical error"
4	"Statement error"
5	"Expression error"
6	"Execution error"
7	"System error"
8	"Fatal error"
9	"Undefined variable error"
10	"Aborted"
11-19	"Undefined server error"
20	"Analytica is uninitialized"
21	"Bad parameter passed"
22	"Value not found in index"
23	"Illegal position in index"

24	"Subscript must be an array of variants"
25	"Subscripts can not be accessed"
26	"Lower bound of subscript array inaccessible"
27	"Upper bound of subscript array inaccessible"
28	"Must specify at least one element in table""
29	"Position specified is out of bounds"
30	"Position does not exist"
31	"Illegal position specified in table"
32	"Index object not found"
33	"Illegal index number specified"
34	"Definition table not found"
35	"Attribute could not be retrieved"
36	“Attribute could not be set”
37	“Could not retrieve result”
38	“Could not get result table”
39	“Module/Model/Script could not be found”
40	“Object could not be created”
41	“Invalid name for object”
42	“Object name already in use”
43	“Current module could not be retrieved”
44	“Module could not be set”
45	“Model could not be created”
46	“Model/Module could not be saved”
47	“Illegal command”
48	“Invalid object class”
49	“There is no model to save”
50	“Safe-array has incorrect size or number of dimensions”
51	“Element position is non-numeric”
52	“Specified name is not an index of the array”

Working with Analytica Models, Modules, and Files

Note: In VB.NET, the Set keyword is no longer necessary when assigning an object as a value. The keyword is necessary in previous versions of Visual Basic, in VBScript, and VBA, and is thus shown in the examples below.

- To create a new model:

```
Ana.CreateModel ("NewModelName")
If Ana.ErrCode = 0 Then
      'Model successfully created
End If
```

  The `CreateModel` method only requires one parameter, a string containing a model name.

- To open an existing Analytica model:

```
Dim FileSpec as String
Dim ModName as String
FileSpec  =  "C:\ ... \Anamodel.ana"
ModName = Ana.OpenModel (FileSpec)
If ModName="" then
      ' Handle Error condition here'
End if
```

  If a model has already been opened, then that model will be closed automatically before the new model is created. If the specified filename is not legal, `OpenModel` will return an empty string. If an empty string is returned, use the `ErrorCode` property of `CAEngine` to determine the cause of the error. Be aware that an `ErrorCode=2` (Warning) is often returned even though the load is successful. For full details as to what has caused an error or warning, use the `OutputBuffer` property of the `CAEngine`. You must specify the backward slash (\) for the path delimiter when using ADE. The forward slash (/) is not yet supported.

- To add a module from a file to the currently open model:

```
Dim ModObj as Object
Dim Merge as Boolean
ModObj = Ana.AddModule (FileSpec, Merge)
```

  The `FileSpec` parameter should contain the path and filename of the module to be included. The `Merge` parameter is a Boolean variable that determines whether preexisting objects with identical names are overwritten. If `Merge=True` then conflicting variables are overwritten. If `Merge=False`, and there are conflicting variables, then the call to `AddModule` will fail.

- To read a script file:

```
Dim RetCode as Boolean
RetCode = Ana.ReadScript (FileSpec)
```

  A script file can contain a list of typescript commands. Upon loading the file, the Engine will execute the commands contained in the file. Errors encountered while running the script file are described in the `ErrorText` property.

- To save a module (i.e., a subset of the current model) in a separate file:

```
RetCode = Ana.SaveModuleFile (ModName, FileSpec)
```

- To save the current model in a file:

```
RetCode = Ana.SaveModel (FileSpec)
```

- To close the current model without saving:

```
Ana.CloseModel
```

  The `CloseModel` method takes and returns no parameters.

ADE Objects

- To create a new `CAObject` object:

```
Dim ObjName, ObjClass as String
Dim Var as Object  ' or as CAObject
ObjName = "NewVariable"
ObjClass = "Chance"
Set Var = Ana.CreateObject (ObjName, ObjClass)
```

  The object name and the class of the object to be created are passed into the `CreateObject` method. Note that an identifier and not the title of the object should be used when giving the object a name. Most Object-related methods use their Identifier attribute, not their Title attribute. ADE can create the following types of objects: Variable, Module, Chance, Constant, Decision, Index, and Objective. Refer to the Analytica Developer's Guide for more information on these object types.

- To delete an Analytica object from a model:

```
RetCode = Ana.DeleteObject (Var)
```

- To set the active module:

```
ObjName = "ModuleToMakeActive"
ObjClass  =  "Module"
Set Var = Ana.CreateObject (ObjName, ObjClass)
Set Ana.CurrentModule = Var
```

  ADE utilizes a hierarchy to order objects. When an object is created, it is created inside the current module. By default, all objects are placed within the top-level module unless you set the `CurrentModule` property.

- To identify the current module:

```
Set Var = Ana.CurrentModule
```

- To obtain a `CAObject` object when you know the name of an Analytica variable (this is probably the most commonly used method in ADE):

```
Dim Var As Object

ObjName = "IdentifierInModel"
Set Var = Ana.GetObjectByName (ObjName)
If Var Is Nothing Then
      'Analytica model associated with Ana
      'does not contain variable with
      'identifier "IdentifierInModel'
End If
```

- All Analytica object attributes, with the exception of the `Name` attribute can be obtained using the `GetAttribute` method:

```
Set AttName = "Units"
UnitsOfVar = Var.GetAttribute (AttName)
```

- To get the Name of an identifier, you should do the following (where `Var` is the `CAObject` associated with the identifier of interest):

```
Name = Var.Name
```

- The `SetAttribute` method is used to set an Analytica object's attributes (except for `Name`):

```
Dim RetCode As Boolean
RetCode = Var.SetAttribute ("definition","A/B")
If RetCode = True Then
      'Attribute Set Correctly
Else
      'Attribute Not Set
End If
```

- To set the `Name` attribute, do the following:

```
Var.Name = "NewNameOfIdentifer"
```

  For the full lists of object attributes see *The Analytica Developer's Guide*, Chapter 3: Objects and their attributes.

Retrieving Results of CAObject

- To evaluate and obtain the result of an object use the `Result` method of `CAObject`:

```
Dim Obj As Object
Dim Result
Set Obj = Ana.GetObjectByName (“ObjectToEvaluate”)
Result = Obj.Result
If Ana.ErrorCode = 0 Then
      ’Result was successfully retrieved
Else
      ’An error occurred
End If
```

  The `Result` property of `CAObject` retrieves, by default, the midpoint result of the object. It will return the result as a variant. This method is convenient for retrieving the results of objects that evaluate to a scalar.

- To evaluate and obtain the result of an object as something other than the midpoint use the `ResultType` property of `CATable`:

```
Dim Obj As Object
Dim Result
Set Obj = Ana.GetObjectByName (“ObjectToEvaluate”)
Obj.ResultType = 1  ’get result as mean
Result = Obj.Result
If Ana.ErrorCode = 0 Then
      ’Result was successfully retrieved as a mean
Else
      ’An error occurred
End If
```

  The `ResultType` property is used to indicate the type of result that `Result` should return. If `ResultType` is set to 0, then the midpoint of the result will be returned by `Result`. If `ResultType` is set to 1, then the mean will be returned by `Result`. If `ResultType` is set to 2, then the probabilistic value of the result will be returned by `Result`. Note that the probabilistic value of the result will always be multi-dimensional, even if the mid and mean are not multi-dimensional. See the next section for a discussion on retrieving multi-dimensional results.

Retrieving Multi-Dimensional Results of CAObject

Before delving into the details of how to obtain results from multi-dimensional objects (tables with one or more dimensions), let us briefly discuss the conceptual model of a table in Analytica.

An Analytica table is composed of the following components:

1. the indexes, which determine the dimensions of the table.
2. the values in the cells of the table.
3. The index labels, which can be used to identify the coordinates of each cell.

The number of indexes determines the dimensionality of the table. So, for example, if a table contains two indexes, then the table is 2-dimensional.

The number of elements in the index determines the actual number of cells in the table. Suppose table T is composed of 2 indexes, I and J. If I has 5 elements (AA, BB, CC, DD, EE) and J has 3 elements (A, B, C), then T is either a 5 X 3 table, or a 3 X 5 table, depending on your perspective.

Determining your perspective of a table is very important when working with ADE. It is up to you to tell ADE how you wish to view the table. So, for example, in the above paragraph, if you tell ADE to use index I first, followed by index J, then element 2,3 would be the element described by position I=B, J=CC. If, however, you tell ADE to use index J first, followed by index I, then element 2,3 would be described by position I=C, J=BB (note that tables in ADE are 1-based; that is, each dimension goes from 1 to N where N is the size of the index). The method called `SetIndexOrder`, described below, allows you to set the order of the indexes for your table, so that you can look at the table in any way you desire.

The ADE methods are very flexible in terms of how you refer to individual elements in the table. You can either refer to the individual elements by their position number or by their label names. So, for example, you can tell ADE to give you the element at position 2,1 (2 along the first index, and 1 along the second index), or you can tell ADE to give you the element described by ‘BB’,’A’ where ‘BB’ and ‘A’ are label names in their respective indexes. The methods most commonly used for these types of transactions (`GetDataByElements` and `GetDataByLabels`) are described below.

As discussed in the previous section, the `Result` and `ResultType` methods are used to evaluate and obtain the result of an object. For objects that evaluate to multi-dimensional results, however, it is often inconvenient to use the `Result` method. After all, the output of the `Result` method for a multi-dimensional result would be a long comma delimited string in the following form:

```
Table Index1...IndexN [Value1, Value2...]
```

Here, `Index1` to `IndexN` are the indexes of the table, and `Value1` to `ValueN` are the values in the table (which are filled in row by row). So, if we wanted to get at a particular element in the table after using the `Result` method, we would have to parse through the comma delimited string, returned from `Result`, in order to get at the element of interest. Fortunately, ADE provides an ADE object of type `CATable` that provides methods to simplify the manipulation of tables.

- To evaluate and obtain the muti-dimensional result of an object use the `ResultType` method of `CAObject`:

```
Dim Obj As Object
Dim TableResult As Object
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableResult = Obj.ResultTable
If Not TableResult Is Nothing Then
      ’Result table was successfully retrieved
Else
      ’An error occurred, or result is scalar
End If
```

The `ResultTable` method of `CAObject` returns an automation object of type `CATable`. `CATable` contains various methods that allow you to set, retrieve, and manipulate individual elements in the table. The first thing that you will, more than likely, want to do after retrieving the `CATable` object, is to set the index order of the result table.

- To set the index order of a `CATable` object, use the `SetIndexOrder` method:

```
Dim Obj As Object
Dim TableResult As Object
Dim IndexOrder (2)
Dim RetValue
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableResult = Obj.ResultTable
```

```
If Not TableResult Is Nothing Then
   'Result table was successfully retrieved
   IndexOrder (1) = "Index2"
   IndexOrder (2) = "Index1"
   RetValue = TableResult.SetIndexOrder (IndexOrder)
   If RetValue = True Then
       'Index Order set successfully
    Else
       'An error occurred in setting index order
    End If
Else
    'An error occurred, or result is scalar
End If
```

The above code assumes that we are manipulating a two-dimensional table. We set the index order of this table so that `Index2` is the first index, and `Index1` is the second index.

- To retrieve an element in a table by index order use the `GetDataByElements` method:

```
Dim Obj As Object
Dim TableResult As Object
Dim IndexOrder (2)
Dim Pos (2)
Dim RetValue
Dim Element
Set Obj = Ana.GetObjectByName ("MultiDimObject")
Set TableResult = Obj.ResultTable
If Not TableResult Is Nothing Then
   'Result table was successfully retrieved
   IndexOrder (1) = "Index2"
   IndexOrder (2) = "Index1"
   RetValue = TableResult.SetIndexOrder (IndexOrder)
   If RetValue = True Then
      'Index Order set successfully
      Pos (1) = 2
      Pos (2) = 1
      Element = TableResult.GetDataByElements (Pos)
      If Ana.ErrorCode = 0 Then
            'element retrieved successfully
      Else
            'an error occurred
      End If
   Else
            'An error occurred in setting index order
   End If
Else
   'An error occurred, or result is scalar
End If
```

The above code uses `GetDataByElements` to retrieve the element at position `Index2=2, Index1=1` and stores the result to `Element`.

- To retrieve an element in a table by index labels use the `GetDataByLabels` method:

```
Dim Obj As Object
Dim TableResult As Object
Dim IndexOrder (2)
Dim Pos (2)
Dim RetValue
Dim Element
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableResult = Obj.ResultTable
If Not TableResult Is Nothing Then
     ’Result table was successfully retrieved
     IndexOrder (1) = “Index2”
     IndexOrder (2) = “Index1”
     RetValue = TableResult.SetIndexOrder (IndexOrder)
     If RetValue = True Then
       ’Index Order set successfully
       Pos (1) = “SomeLabelInIndex2”
       Pos (2) = “SomeLabelInIndex1”
       Element = TableResult.GetDataByLabels (Pos)
       If Ana.ErrorCode = 0 Then
             ’element retrieved successfully
       Else
             ’an error occurred
       End If
     Else
       ’An error occurred in setting index order
     End If
Else
     ’An error occurred, or result is scalar
End If
```

The above code uses `GetDataByLabels` to retrieve the element at position `Index2=”SomeLabelInIndex2”, Index1=”SomeLabelInIndex1”` and stores the result to `Element`.

- To retrieve the whole table into a Visual Basic array in one call use the `GetSafeArray` method:

```
Dim Obj As Object
Dim TableResult As Object
Dim IndexOrder (2)
Dim Pos (2)
Dim RetValue
Dim TheWholeTable
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableResult = Obj.ResultTable
```

```
If Not TableResult Is Nothing Then
     'Result table was successfully retrieved
     IndexOrder (1) = "Index2"
     IndexOrder (2) = "Index1"
     RetValue = TableResult.SetIndexOrder (IndexOrder)
     If RetValue = True Then
       'Index Order set successfully
       Pos (1) = "SomeLabelInIndex2"
       Pos (2) = "SomeLabelInIndex1"
       TheWholeTable = TableResult.GetSafeArray
       If Ana.ErrorCode = 0 Then
             'table retrieved successfully
       Else
             'an error occurred
       End If
     Else
       'An error occurred in setting index order
     End If
Else
     'An error occurred, or result is scalar
End If
```

The above code uses `GetSafeArray` to store the entire table in `TheWholeTable`. The elements of each dimension associated with the table returned from `GetSafeArray` are indexed 1 to N, where N is the length of the dimension.

- To determine the number of dimensions of the table, use the `NumDims` property:

```
Dim Obj As Object
Dim NumDimensions
Dim TableResult As Object
Set Obj = Ana.GetObjectByName ("MultiDimObject")
Set TableResult = Obj.ResultTable
NumDimensions = TableResult.NumDims
```

- To get the index names associated with the table, use the `IndexNames method:`

```
Dim Obj As Object
Dim NumDimensions
Dim TableResult As Object
Dim I As Integer
Dim CurIndexName As String
Set Obj = Ana.GetObjectByName ("MultiDimObject")
Set TableResult = Obj.ResultTable
NumDimensions = TableResult.NumDims
For I = 1 To NumDimensions
      CurIndexName = TableResult.IndexNames (I)
      MsgBox "Current index is " & CurIndexName
Next I
```

`IndexNames` will return the index names in the order specified by `SetIndexOrder`. If `SetIndexOrder` has not been used for the `CATable`, then the default order of the indexes will be returned.

- To get the `CAIndex` objects associated with your table, use the `GetIndexObject` method of `CATable`:

```
Dim Obj As Object
Dim NumDimensions
Dim TableResult As Object
Dim CurIndexName As String
Dim IndexObj As Object
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableResult = Obj.ResultTable
NumDimensions = TableResult.NumDims

CurIndexName = TableResult.IndexNames (NumDimensions)
Set IndexObj = TableResult.GetIndexObject
(CurIndexName)
```

The above example retrieved the last `CAIndex` object, with respect to the index order, from the table. The `CAIndex` object provides properties and methods that allow you to obtain information about the respective index.

- To get the number of elements in the index, use the `IndexElements` property:

```
Dim Obj As Object
Dim NumDimensions
Dim NumElsInIndex
Dim TableResult As Object
Dim CurIndexName As String
Dim IndexObj As Object
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableResult = Obj.ResultTable
NumDimensions = TableResult.NumDims
CurIndexName = TableResult.IndexNames (NumDimensions)
Set IndexObj = TableResult.GetIndexObject
(CurIndexName)
NumElsInIndex = IndexObj.IndexElements
```

- To get an index label at the specified position in the index, use the `GetValueByNumber` method:

```
Dim Obj As Object
Dim NumDimensions
Dim NumElsInIndex
Dim TableResult As Object
Dim CurIndexName As String
Dim IndexObj As Object
```

```
      Dim I As Integer
      Dim Str As String

      Set Obj = Ana.GetObjectByName (“MultiDimObject”)
      Set TableResult = Obj.ResultTable
      NumDimensions = TableResult.NumDims
      CurIndexName = TableResult.IndexNames (NumDimensions)
      Set IndexObj = TableResult.GetIndexObject
(CurIndexName)
      NumElsInIndex = IndexObj.IndexElements
      Str = “The elements in the index are: ” & vbCrLf
      For I=1 To NumElsInIndex
            Str = Str & IndexObj.GetValueByNumber (I) & “ “
      Next I
      MsgBox Str
```

- To get at the position of an index label in an index, use the `GetNumberByValue` method:

```
      Dim Obj As Object
      Dim NumDimensions
      Dim TableResult As Object
      Dim CurIndexName As String
      Dim IndexObj As Object
      Dim I As Integer
      Dim IndexPosition As Integer

      Set Obj = Ana.GetObjectByName (“MultiDimObject”)
      Set TableResult = Obj.ResultTable
      NumDimensions = TableResult.NumDims
      CurIndexName = TableResult.IndexNames (NumDimensions)
      Set IndexObj = TableResult.GetIndexObject
(CurIndexName)
      IndexPosition = IndexObj.GetNumberByValue
(“SomeIndexLabel”)
      If Ana.ErrorCode = 0 Then
            ’the index position was successfully retrieved
      Else
            ’an error occurred
      End If
```

Often times, it is difficult to determine whether the evaluation of an object will return a multi-dimensional result or a scalar. What you should do in this scenario is to first call `ResultTable` on the `CAObject` of interest. If the call succeeds, then you have a multi-dimensional result. If `ResultTable` fails, call `Result`. If `Result` succeeds in this scenario, then the result is a scalar. Otherwise, there was an error in the evaluation of the object. So, the following code might be used:

```
Dim Obj As Object
Dim TableResult As Object
Dim ScalarResult

Set Obj = Ana.GetObjectByName (“SomeObject”)
Set TableResult = Obj.ResultTable
If TableResult Is Nothing Then
      ScalarResult = Obj.Result
      If Ana.ErrorCode = 0 Then
            ’you have a scalar result
      Else
            ’an error occurred
      End If
Else
      ’you have a multi-dimensional result
End If
```

Creating Tables and Setting Values In Tables

Up until this point, we talked about how to retrieve elements from result tables. We can use the same exact methods described above to retrieve values in definition tables. A definition table, as the name suggests, is when the definition of an object is defined to be a table. There are a couple of interesting things to point out at this point:

1. Just because `ResultTable` returns a table, it does not mean that the definition of the object is a definition table. For example, an object could be defined to be a function call which, when evaluated, evaluates to a table. But a function call is not a definition table.
2. Each cell in a definition table contains an expression. Therefore, strings must be quoted (i.e. 'samplestring'). This is not the case for result tables, since all cells in a result table contain either a string or a number.
3. An object defined to be a definition table would not necessarily produce the same table when `ResultTable` is called. After all, the definition table can be defined to be an array of identifiers. When `ResultTable` is called, each identifier's result will be evaluated, and a new table will be produced which would be different than the definition table. If identifiers evaluate to arrays, the result table may have more dimensions than the definition table.

- To get the definition table of an object as a `CATable`, use the `DefTable` method of `CAObject`:

```
Dim Obj As Object
Dim TableDef As Object
Set Obj = Ana.GetObjectByName ("MultiDimObject")
Set TableDef = Obj.DefTable
If Not TableDef Is Nothing Then
     'Definition table was successfully retrieved
Else
     'An error occurred, or definition is not a table
End If
```

Once the definition table is retrieved, we can use all the same methods described in the above section (`GetDataByElements`, `GetDataByLabels`, `SetIndexOrder`, etc.) to retrieve elements in the table and to obtain information about the indexes in the table. We can also use the same method that we used above in

determining whether the result of the object was multi-dimensional or scalar to determine whether the definition of the object is a table or scalar:

```
Dim Obj As Object
Dim TableDefinition As Object
Dim ScalarDefinition

Set Obj = Ana.GetObjectByName (“SomeObject”)
Set TableDefinition = Obj.DefTable
If TableDefinition Is Nothing Then
   ScalarDefinition = Obj.GetAttribute (“definition”)
   If Ana.ErrorCode = 0 Then
      ’you have a scalar definition
   Else
      ’an error occurred
   End If
Else
   ’you have a table definition
End If
```

- To set an element in a table by index order use the `SetDataByElements` method of `CATable`:

```
Dim Obj As Object
Dim TableDef As Object
Dim IndexOrder (2)
Dim Pos (2)
Dim RetValue
Dim Element
Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableDef = Obj.DefTable
If Not TableDef Is Nothing Then
   ’Definition table was successfully retrieved
   IndexOrder (1) = “Index2”
   IndexOrder (2) = “Index1”
   RetValue = TableDef.SetIndexOrder (IndexOrder)
   If RetValue = True Then
     ’Index Order set successfully
     Pos (1) = 2
     Pos (2) = 1
     Element = “’ABC’”
     RetValue=TableDef.SetDataByElements(Element,Pos)
     If RetValue = True Then
       ’element successfully set
       RetValue = TableDef.Update
       If RetValue = True Then
             ‘model successfully updated
       Else
             ‘error updating def in model
       End If
```

```
    Else
      'an error occurred
    End If
  Else
    'An error occurred in setting index order
  End If
Else
     'An error occurred, or definition is scalar
End If
```

The above code uses `SetDataByElements` to set the element at position `Index2=2, Index1=1` to `Element`. Note the use of the quotes around ABC. Here, since ABC is single quoted, we are putting the string "ABC" in the table. If we instead set `Element` to "ABC", then the expression ABC would be placed in the table. In the latter case, ABC would, more than likely, be a variable. If an identifier, ABC, did not exist in the model, then an error would have occurred while trying to set the element in the latter case. The code then used `Update` to update the model with the new definition. It is important to note that the model containing the object will not be updated until `Update` is called. Therefore, if `Update` is not called, and the result of a node that depends on this object is later calculated, the old definition of this object will still be used. The other important thing to note is that `Update` functions very differently for result tables as it does for definition tables. For result tables, `Update` will retrieve the result from the specified object again. It will, therefore, overwrite any changes that were made to the object using `SetDataByElements` and `SetDataByLabels`.

- To set an element in a table by index labels use the `SetDataByLabels` method of `CATable`:

```
Dim Obj As Object
Dim TabDef As Object
Dim IndexOrder (2)
Dim Pos (2)
Dim RetValue
Dim Element
Set Obj = Ana.GetObjectByName ("MultiDimObject")
Set TabDef = Obj.DefTable
If Not TabDef Is Nothing Then
   'Definition table was successfully retrieved
   IndexOrder (1) = "Index2"
   IndexOrder (2) = "Index1"
   RetValue = TabDef.SetIndexOrder (IndexOrder)
   If RetValue = True Then
      'Index Order set successfully
```

```
      Pos (1) = “SomeLabelInIndex2”
      Pos (2) = “SomeLabelInIndex1”
      Element = “’ABC’”
      RetValue=TabDef.SetDataByLabels(Element,Pos)
      If RetValue = True Then
            ’element set successfully
            RetValue = TabDef.Update
            If RetValue = True Then
                  ’model successfully updated
            Else
                  ’an error occurred
            End If
      Else
            ’an error occurred
      End If
   Else
      ’An error occurred in setting index order
   End If
Else
      ’An error occurred, or definition is scalar
End If
```

The above code uses `SetDataByLabels` to set the element at position `Index2=”SomeLabelInIndex2”, Index1=”SomeLabelInIndex1”` to `Element`.

- To set the whole table in one call, use `PutSafeArray` and `Update`:

```
Dim Obj As Object
Dim TableResult, TableDef As Object
Dim RetValue
Dim TheWholeTable

Set Obj = Ana.GetObjectByName (“MultiDimObject”)
Set TableDef = Obj.DefTable
Set TableResult = Obj.ResultTable
TheWholeTable = TableResult.GetSafeArray
‘make changes to TheWholeTable
…
RetValue = TableDef.PutSafeArray (TheWholeTable)
If RetValue = True Then
      ’table successfully put
      RetValue = TableDef.Update
      If RetValue = True Then
            ’model successfully updated
      End If
End If
```

- To create a whole table from scratch, use `CreateDefTable`:

```
Dim Obj As Object
Dim RetValue
Dim IndexLabs (1 To 2) As Variant

Set Obj = Ana.CreateObject (“MyNewTable”, “Variable”)
IndexLabs (1) = “I”
IndexLabs (2) = “J”
RetValue = Obj.CreateDefTable (IndexLabs)
If RetValue = True Then
      ’a table indexed by I and J has successfully
      ’been created.  We are assuming that I and J
      ’already exist
Else
      ’an error occurred when creating the table
End If
```

  The above code created a definition table indexed by `I` and `J`. The table is dimensioned according to the size of `I` and `J`. All the cells in the table are initially set to 0. The user can then call `DefTable`, and then use `SetIndexOrder`, `SetDataByElements`, `SetDataByLabels`, `PutSafeArray`, and `Update` to put values into the table. Note that the function `CreateDefTable` is very rarely used in an ADE program. After all, it is much easier to create an object in Analytica than it is in ADE.

Analytica Decision Engine Server Class Reference

Class CAEngine

Properties:

Command

Description: Sets a typescript language command for execution by the Analytica Decision Engine Automation Server. The engine is directed to execute the command using the Send method. For the list of typescript commands see the *Analytica Scripting Guide.*
Data type: string
Access: read/write
Usage: `Ana.Command = "value obj1"`
API Errors: None

CurrentModule

Description: Currently opened module.
Data type: CAObject
Access: read/write
Usage:

```
Dim Obj As Object
Set Obj = Ana.CurrentModule
```

Remarks: User must set this property to define a parent module for new objects. Setting `CurrentModule = Nothing` means that no module is opened, all new objects will be created in the top-level diagram of the currently opened model.

API Errors: 39- " Module could not be found"
44- " Module could not be set"

ErrorCode

Description: Returns the error code generated by the last communication with the Analytica Decision Engine Server. The property `ErrorCode` should be checked after setting and retrieving critical `CAEngine` properties and calling `CAEngine` methods. An `ErrorCode` value of zero indicates successful completion of the last action by the Analytica Decision Engine Server.

Data type: integer

Access: read

Usage:

```
Dim x As Integer
x = Ana.ErrorCode
```

Error Codes:
0: "OK"
1: "Unimplemented"
2: "Warning"
3: "Lexical error"
4: "Statement error"
5: "Expression error"
6: "Execution error"
7: "System error"
8: "Fatal error"
9: "Undefined variable error"
10: "Aborted"
20: "Analytica is uninitialized"
21: "Bad parameter passed"
22: "Value not found in index"
23: "Illegal position in index"
24: "Subscripts must be an array of variants"
25: "Subscripts cannot be accessed"
26: "Lower bound of subscript array inaccessible"
27: "Upper bound of subscript array inaccessible"
28: "Must specify at least one element in table"
29: "Position specified is out of bounds"
30: "Position does not exist"
31: "Illegal position specified in table"
32: "Index object not found"
33: "Illegal index number specified"
34: "Definition table not found"
35: "Attribute could not be retrieved"
36: "Attribute could not be set"
37: "Could not retrieve result"
38: "Could not get result table"
39: "Module/Model/Script could not be found"
40: "Object could not be created"
41: "Invalid name for object"
42: "Object name already in use"

43: "Current module could not be retrieved"
44: "Module could not be set"
45: "Model could not be created"
46: "Model/Module could not be saved"
47: "Illegal command"

ErrorText
Description: short text explanation of error from ErrorCode
Data type: string
Usage:

```
Dim x As String
x = Ana.ErrorText
```

Access: read

Log
Description: contains a record of communications between the client and the Analytica Decision Engine Server when using typescript.
Remarks: A record of communications will not be logged unless the Photo property is turned on.
Data type: string
Usage:

```
Dim x As String
x = Ana.Log
```

Access: read

OutputBuffer
Description: A text string buffer that contains the result of the last typescript (i.e., using the Command property and Send method) interaction with the Analytica Decision Engine Server.
Data type: string
Usage:

```
Dim x As String
x = Ana.OutputBuffer
```

Access: read

Photo

Description: The user may review all typescript communications between the client and the Analytica Decision Engine Server by setting the `Photo` property to `True` (and by calling the `Log` method of CAEngine).
Data type: Boolean
Access: read/write
Usage: `Ana.Photo = True`
Remarks: setting Photo property to True slows down computation speed of the Engine.

Methods:

AddModule

Description: Adds a module to an already opened/created model
Parameters: FileSpec - string
Merge – Boolean.
Return Value: ModuleName - string
Usage: `ModName = Ana.AddModule (“C:\MYMOD\MYMOD.ANA”, True)`
Remarks: Where the module is being added depends on the value of the CurrentModule property. Merge currently has no effect and should be set to True.
API Errors: 39- "Module could not be found"

CloseModel

Description: Closes the model
Usage: `Ana.CloseModel`

CreateObject

Description: Creates a new CAObject object and a new Analytica object in the CurrentModule
Parameters: ObjName - string
ObjClass – string
Return Value: CAObject
Usage:

```
Dim obj As CAObject
Set obj = Ana.CreateObject (“NewVar”,”Chance”)
```

Remarks: ObjClass can be one of the following values: Decision, Variable, Chance, Constant, Index, Module, Objective, Determ, Alias, or Formnode.

API Errors: 40- "Object could not be created"
41- "Invalid name for object"
42 - "Object name already in use"
48 – “Invalid object class”

CreateModel
Description: Creates a new Analytica model
Parameters: ModelName - string
Return Value: Boolean (success or failure)
Usage: `boolval = Ana.CreateModel (“MyNewModel”)`
API Errors: 45- "Model could not be created"

DeleteObject
Description: Deletes the specified CAObject from the Current Model
Parameters: Var – CAObject
Usage:

```
Dim Obj As CAObject
Set Obj = Ana.GetObjectByName(“ObjToDelete”)
Ana.DeleteObject (Obj)
```

API Errors: 41- "Invalid object"

ResetError
Description: Resets the error code, error text string associated with the error code, and the output buffer to default values. This function is normally used internally, but could be useful in other circumstances as well.
Usage `Ana.ResetError`

GetObjectByName
Description: Returns an object of type CAObject for an existing Analytica object.
Parameters: ObjName - string
Return Value: CAObject
Usage:

```
Dim Obj As CAObject
Set Obj = Ana.GetObjectByName (“MyObject”)
```

API Errors: 41- "Invalid name for object"

OpenModel
Description: Reads a model from a disk file and opens it as the current model.
Parameters: FileSpec- string (the filename containing the model).
Return Value: ModelName – string (actual model name).
Usage: `modName = Ana.OpenModel (“C:\TMP\MYMODEL.ANA”)`
Remarks: Failure should be detected by checking whether the return value is “”, not by checking for a zero ErrorCode. It is possible that some errors or warnings may occur during loading, and will thus be reflected in the ErrorCode, ErrorText, and OutputBuffer properties, even though the load was successful.

API Errors: 2– Warning (but load was successfully completed).
3– Lexical error (load was only partially successful)
4– Statement error (load was only partially successful)
39- "Model could not be found"

ReadScript
Description: Reads an Analytica script file and executes it immediately.
Parameters: FileSpec – string
Usage: `Ana.ReadScript ("C:\TMP\SCRIPT.MOD")`
API Errors: 39- "Script file could not be found"

SaveModel
Description: Saves the model to a file
Parameters: FileSpec – string
Usage: `Ana.SaveModel("C:\TMP\CHANGES.ANA")`
API Errors: 46- "Model could not be saved"
49- "There is no model to save"

SaveModuleFile
Description: Saves a module to a file (i.e., to a file separate from the file in which the current model is saved).
Parameters: ModuleName – string
FileSpec - string
Return Value: Boolean (success or failure)
Usage: `b = Ana.SaveModuleFile("ModName", "C:\TMP\NEWMOD.ANA")`
API Errors: 41 – "Invalid name for object"
46- "Module could not be saved"

Send
Description: Sends the Command property to the Analytica Decision Engine Server.
Return Value: Boolean (success or failure)
API Errors: 1- "Unimplemented"
2- "Warning"
3- "Lexical error"
4- "Statement error"
5- "Expression error"
6- "Execution error"
7- "System error"
8- "Fatal error"
9- "Undefined variable error"
10- "Aborted"

Class CAObject

Properties:

ClassType
Description: Contains the type of the Analytica object.
Data type: string
Access: read/write
Usage `classType = CAObject.ClassType`
Remarks: ADE currently supports the following types of Analytica objects: Decision, Chance, Constant, Index, Module, and Variable.

DefTable
Description: An object of class CATable containing the input table for the specified Analytica object.
Data type: CATable
Access: read/write
Usage:

```
Dim CATable As Object
Set CATable = CAObject.DefTable
```

Remarks: returns Nothing if the object's definition is not a valid Analytica table
API Errors: 34- "Definition table not found"

Name
Description: Contains the name given to the Analytica object.
Data type: string
Access: read/write
Usage: `CAObject.Name = "NewName"`
API Errors: 41- "Invalid name for object"

Result
Description: Contains the evaluated result value of an object. Accessing this property will cause the Analytica Decision Engine Server to evaluate the specified object. Multidimensional results are expressed in a string of comma-delimited elements.
Data type: variant
Access: read
Usage:

```
Dim x
X = CAObject.Result
```

API Errors: 37- "Could not retrieve result"

ResultTable

Description: An object of class CATable containing the multidimensional result value of the specified Analytica object.
Data type: CATable
Access:read
Usage:

```
Dim CATable As Object
Set CATable = CAObject.ResultTable
```

Remarks: returns Nothing if object's result is not a valid Analytica table
API Errors: 38- "Could not get result table"

ResultType

Description: Indicates the type of result obtained using the Result property:
0 - mid value (default)
1 - mean value
2 - prob value
Data type: integer
Access: read/write
Usage: `CAObject.ResultType = 1`

Methods:

CreateDefTable

Description: This method creates an input table object in the definition attribute of the specified (and pre-existing) Analytica object. The IndexList parameter must contain an array of identifiers for the Table's (pre-existing) indexes (this parameter is identical in form to the IndexNames property of class CATable). The number of elements in the array passed to CreateDefTable determines the number of dimensions of the table. If the table is indexed by itself, "Self" should be one of the entries in the array. Initially, the input table object's array will be filled with null elements, which can be changed using the SetDataByElements and SetDataByLabels methods of the class CATable.
Parameters: IndexList - array of strings
Return Value: Boolean (success or failure)
Usage: `Var.CreateDefTable (IndexList)`
API Errors: 25- "Subscripts cannot be accessed"
26- "Lower bound of subscript array inaccessible"
27- "Upper bound of subscript array inaccessible"
28- "Must specify at least one element in table"
32- "Index object not found"

GetAttribute
Description: Gets an object attribute value
Parameters: AttributeName: - string
Return Value: variant
Usage: `X = CAObject.GetAttribute(“definition”)`
API Errors: 35- "Attribute could not be retrieved"

SetAttribute
Description: Sets an object attribute value
Parameters: AttributeName: - string
AttributeValue: - Variant
Return Value: Boolean (success or failure)
Usage: `bool = CAObject.SetAttribute (“definition”,”A/B”)`
API Errors: 36- "Attribute could not be set"

Class CATable

Properties:

IndexNames
Description: This property allows access to the indexes of the table. The order of the indexes should be adhered to when using the GetDataBy... SetDataBy... methods to retrieve or set selected table values. If you wish to change this order, use the SetIndexOrder method.
Data type: string array with dimension from 1 to object’s NumDims property
Access: read
Usage: `String_array (k) = Var.DefTable.IndexNames (k)`
API Errors: 33- "Illegal index number specified"

NumDims
Description: This property shows the number of dimensions of a multi-dimensional object.
Data type: integer
Access: read
Usage: `x = CATable.NumDims`

TableType
Description: This property holds the type of the table (“D” for a definition table, and “V” for a result table)
Data type: string
Access: read
Usage: `x = CATable.TableType`

Methods:

GetDataByLabels

Description: Retrieves the value of an input table cell according to index labels.
Return Value: variant
Parameters: values of indexes (Variant); the number of elements in the Variant must be equal to NumDims.
Usage:

```
IndexLabs (1) = 3
IndexLabs (2) = “green”
W = Var.DefTable.GetDataByLabels (IndexLabs)
```

Or

```
IndexLabs (1) = 3
IndexLabs (2) = “green”
W = Var.ResultTable.GetDataByLabels (IndexLabs)
```

If the table is one dimensional, then an array is not needed:

```
W = Var.ResultTable.GetDataByLabels (“green”)
```

API Errors: 24- "Subscripts must be an array of variants"
25- "Subscripts cannot be accessed"
26- "Lower bound of subscript array inaccessible"
27- "Upper bound of subscript array inaccessible"
28- "Must specify at least one element in table"
30- "Position does not exist"

GetDataByElements

Description: Retrieves the value of input table cell according to index values.
Return Value: variant
Parameters: index values (Variant), number of elements in the variant must be equal to NumDims.
Usage:

```
IndexPtrs (1) = 1
IndexPtrs (2) = 2
W = Var.DefTable.GetDataByElements (IndexPtrs)
```

Or

```
IndexPtrs (1) = 1
IndexPtrs (2) = 2
W = Var.ResultTable.GetDataByElements (IndexPtrs)
```

If the table is one dimensional, then an array is not needed:

```
W = Var.ResultTable.GetDataByElements (1)
```

API Errors: 24- "Subscripts must be an array of variants"
25- "Subscripts cannot be accessed"
26- "Lower bound of subscript array inaccessible"
27- "Upper bound of subscript array inaccessible"
28- "Must specify at least one element in table"
29- "Position specified is out of bounds"
30- "Position does not exist"
31- "Illegal Position Specified In Table"

GetIndexObject

Description: Retrieves an index object by its name
Return Value: CAIndex
Parameters: index name: string
Usage:

```
dim AI as Object
Set AI = AObj.GetIndexObject (IndexName)
```

Remarks: If ObjName is not valid the method returns Nothing.
API Errors: 32- "Index object not found"

GetSafeArray

Description: Retrieves the CATable as a safe array (i.e. Visual Basic array). The ordering of the dimensions is controlled by the SetIndexOrder method. The elements of each dimension are indexed 1 to N, where N is the amount of elements in the index.
Return Value: Array
Usage:

```
dim Var As Object
Dim curTable
curTable = Var.GetSafeArray
```

API Errors: None.

PutSafeArray

Description: Replaces the current table represented by this object with another table of the same dimensions.
Return Value: CATable
Parameters: the table (Visual Basic Array) that will replace the current table)
Usage:

```
Dim Var As Object
Dim TheArray

TheArray = Var.GetSafeArray
Var.PutSafeArray (TheArray)
```

API Errors: 24- "Subscripts must be an array of variants"
50- "Safe-array has incorrect size or number of dimensions"

SetDataByLabels

Description: Sets the value of an input table cell according to its index labels.
Return Value: Boolean (success or failure)
Parameters: cell value (Variant), values of indexes (Variant), number of elements in this variant must be equal to NumDims.
Usage:

```
IndexVals (1) = 3
IndexVals (2) = ‘green’
RetVal= Var.DefTable.SetDataByLabels (W, IndexVals)
```

If the table is one-dimensional, then an array is not needed:

```
W = Var.DefTable.SetDataByLabels (W, “green”)
```

API Errors: 24- "Subscripts must be an array of variants"
25- "Subscripts cannot be accessed"
26- "Lower bound of subscript array inaccessible"
27- "Upper bound of subscript array inaccessible"
28- "Must specify at least one element in table"
30- "Position does not exist"
31- "Illegal position specified in table"

SetDataByElements

Description: Sets the value of input table cell according to index values.
Return Value: Boolean (success or failure)
Parameters: cell value (Variant), index values (Variant), number of elements in this variant must be equal to NumDims.
Usage:

```
IndexPtrs (1) = 1
IndexPtrs (2) = 2
RetVal = Var.DefTable.SetDataByElements (W, IndexPtrs)
```

If the table is one-dimensional, then an array is not needed:

```
W = Var.DefTable.SetDataByElements (W, 1)
```

API Errors: 24- "Subscripts must be an array of variants"
25- "Subscripts cannot be accessed"
26- "Lower bound of subscript array inaccessible"
27- "Upper bound of subscript array inaccessible"
28- "Must specify at least one element in table"
29- "Position specified is out of bounds"
31- "Illegal position specified in table"
51- “Element position is non-numeric”

SetIndexOrder

Description: Sets the order of the indexes in the table. The order of the indexes determines the order in which SetDataByElements, SetDataByLabels, GetDataByElements, and GetDataByLabels will access a cell in a table.
Return Value: Boolean (success or failure)
Parameters: index names (array of strings). The number of elements in this array must be equal to NumDims.
Usage:

```
IndexVals (1) = ‘X’
IndexVals (2) = ‘Y’
RetVal = Var.DefTable.SetIndexOrder (W, IndexVals)
```

API Errors: 24- “Subscripts must be an array of variants”
26- "Lower bound of subscript array inaccessible"
27- "Upper bound of subscript array inaccessible"
28- "Must specify at least one element in array"
52- "Specified name is not an index of the array"

Update

Description:	For Definition Tables: Updates an existing input table in the definition attribute of an Analytica object. Use this method after setting one or more SetDataBy… methods to direct the API to send the new table data to the Analytica Decision Engine Server. For Result Tables: Retrieves an updated version of the result table from the Analytica Decision Engine Server.
Return Value:	Boolean (success or failure)
Usage:	`Var.DefTable.Update`
Remarks:	Use the CreateDefTable method to replace the current definition attribute of an Analytica object with an input table.
API Errors:	None.

Class CAIndex

Properties:

IndexElements
Description: Returns the number of elements in the index.
Data type: integer
Access: read
Usage: `x = CAIndex.IndexElements`

Name
Description: Contains the name given to the Analytica index.
Data type: string
Access: read/write
Usage: `theName = CAIndex.Name`
API Errors: 41- "Invalid name for object"

Methods:

GetNumberByValue
Description: returns the position of an index label in an index.
Parameters: `Value` - variant
Return Value: integer
Usage: `n = Anindex.GetNumberByValue (Value)`
API Errors: 22- "Value not found in index"

GetValueByNumber
Description: returns the index label at the specified position in the index.
Parameters: Number - integer
Return Value: variant
Usage: `W = AnIndex.GetValueByNumber (Number)`
Remarks:
API Errors: 23- " Illegal position in index"