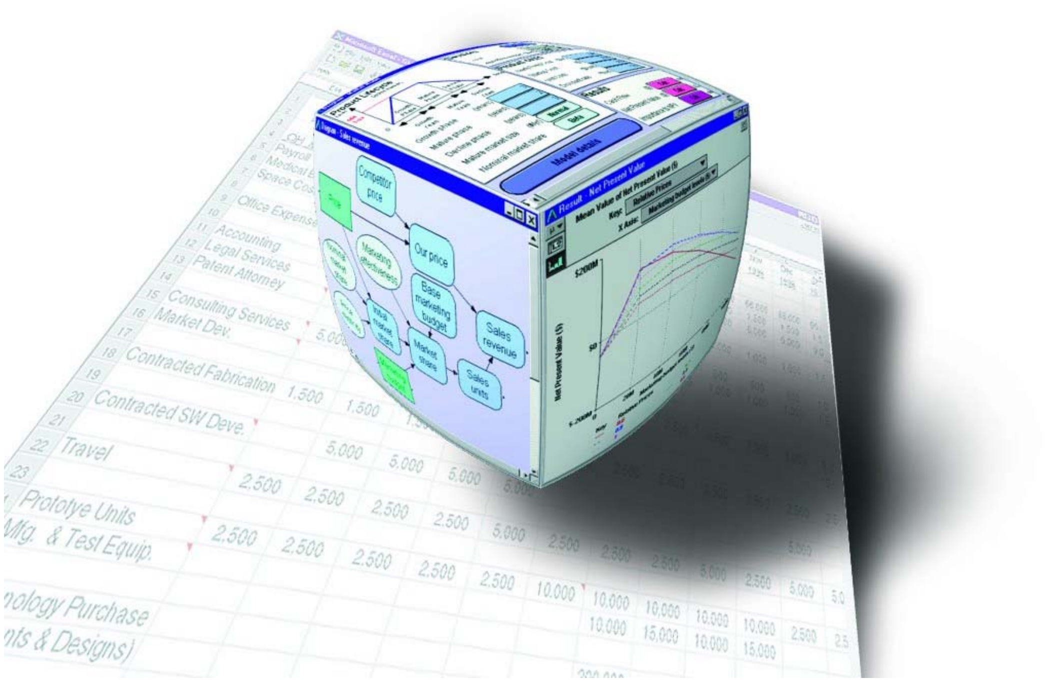
Analytica® Decision Engine for Windows



Scripting Guide

Release 3.0

November, 2003

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

Analytica Scripting Guide was written by Richard Sonnenblick and Hugh Silin

Contents

1. Introduction

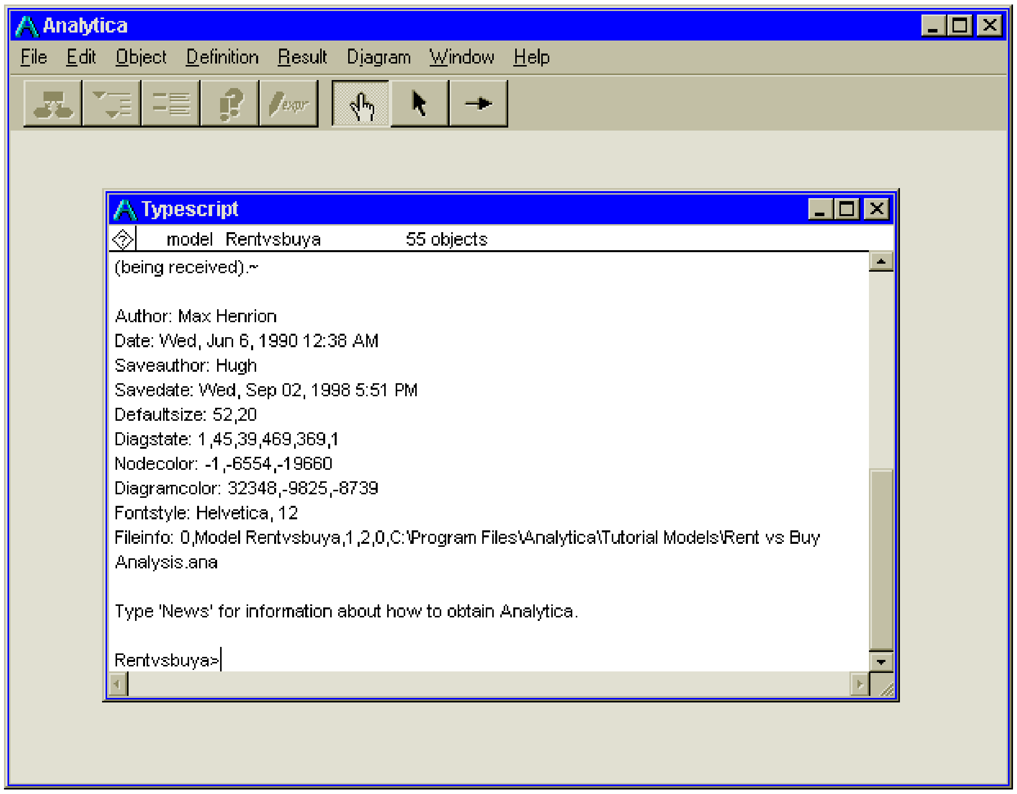
Analytica contains a powerful scripting language for retrieving and modifying information contained in Analytica models. The scripting language is fully accessible from within Analytica (through the use of Analytica's Typescript Window or by using the Command and Send properties of the CAEngine interface of the Analytica Decision Engine (ADE).

The Typescript Window

Hidden underneath the user-friendly graphical user interface of Analytica, your model is a network of objects with associated attributes, and a rich command language with which to manipulate them. The Typescript window provides access to this object-oriented system, from directly within the Analytica application. From this window, you can enter commands and see the results immediately. Later on, as you are developing a program with ADE, the very same interactive language will be used through the Command and Send properties of ADE's CAEngine interface.

To Open the Typescript window from within Analytica

To open the typescript window from within Analytica, hold down the Control key and the single-quote key simultaneously. The following typescript window will appear:


Analytica
File Edit Object Definition Result Diagram Window Help
Typescript
model Rentvsbuya 55 objects

```
(being received).~

Author: Max Henrion
Date: Wed, Jun 6, 1990 12:38 AM
Saveauthor: Hugh
Savedate: Wed, Sep 02, 1998 5:51 PM
Defaultsize: 52,20
Diagstate: 1,45,39,469,369,1
Nodecolor: -1,-6554,-19660
Diagramcolor: 32348,-9825,-8739
Fontstyle: Helvetica, 12
Fileinfo: 0,Model Rentvsbuya,1,2,0,C:\Program Files\Analytica\Tutorial Models\Rent vs Buy
Analysis.ana

Type 'News' for information about how to obtain Analytica.

Rentvsbuya>
```

The Typescript window, similar to a TTY console, displays all commands entered in Analytica, and the corresponding output. At the bottom of the window is a command prompt. When you enter commands at this prompt, the command and its response are added to the bottom of the window.

To Use Typescript from a program making use of ADE

Using ADE, one can access Analytica's typescript language through the use of the Command and Send properties of the CAEngine interface. So, if one wanted to get the result of the Mpy variable in the currently opened model, they would do the following:

```
Ana.Command = "value Mpy"

Ana.Send

TheResult = Ana.OutputBuffer
```

Note that since the result returned by the OutputBuffer property of CAEngine is of the same form that you will see in the typescript window, you may have to parse through the tabs and newlines in the result in order to get at the meaningful part of the result. All other methods of the Analytica Decision Engine, other than Command and Send, automatically parse the result for you.

For more information about using typescript commands with the Analytica Decision Engine, please consult the Analytica Decision Engine for Windows Developers Guide.

Conventions used in this guide

This guide will show you many examples for communicating with the Analytica application using scripts. For consistency, most examples will be in the form that you can use in the Typescript window.

When commands and syntax are presented in this document, you will see examples like this:

```
Example> show mpy
variable Mpy
Title: miles per year
Units: Miles/year
Description: Annual mileage driven
Definition: 12K
Nodelocation: 208,48
Nodesize: 52,24
Example>
```

The part “Example>” is the prompt. The bold text represents text that you type at this prompt. The text following the command and before the next prompt is the response.

Scripts and inter-application communications (IAC, described later in this document) are protocols for performing the same commands you would enter at the prompt, and the output returned through communications are the same as what Analytica would return in the Typescript window. As in the example above, you would execute a script in Analytica, or send a script to Analytica specified as:

```
show mpy
```

and the response would return as:

```
variable Mpy
Title: miles per year
Units: Miles/year
Description: Annual mileage driven
Definition: 12K
Nodelocation: 208,48
Nodesize: 52,24
```

2. Objects and Their Attributes

The entities known to Analytica, such as commands, models, and variables, are called **Objects**. Each object has a unique **Identifier**, and a set of properties known as **Attributes**. For example, many objects have Title, Units, and Description attributes. Every object belongs to a general category, called a **Class**. Object classes include variables, functions, and models (among others).

Creating New Objects

You can create new objects of the following base classes: variable, Function, attribute, and model. Some classes form a hierarchy. There are additional variable and model classes. The usual way to create a new object is to specify the Object class, followed by a unique identifier. For example, in the Typescript window:

```
Example>variable life
```

▲ **Note** Analytica will not be able to evaluate variables in the model until it understands the definitions of all of the objects it contains.

When creating a new object, you should also specify the following user**-specified attributes** by default:

Variable	Title, Units, Description, Definition.
Module	Title, Description, Author.
Attribute	Title, Description.
Function	Title, Description, Definition, Parameters.
Button	Title, Description, Script.

Attributes That are Longer Than One Line

User-specified attributes may be longer than one line. To **continue** a Definition, Description, or other attribute over several lines, you place a tilde ("**~**") or continuation character (option-L, "¬") at the end of each line to be continued (Analytica will finish the Description or Definition when it receives a line that does not end with a tilde).

Analytica will also allow you to continue an attribute over several lines if you leave an open parenthesis on the first line of the attribute. When you close the parentheses, Analytica will assume that you have finished typing in the attribute.

▲ **Note** When entering attributes using the Analytica user interface (the Object window, Diagram window etc.), you do not need to specify a continuation character.

In ADE, it is best to build a large string, and then pass that string directly to the Command property, without using the continuation characters. For example,

```
s = "very long string….."
Ana.Command = "title: " & s
Ana.Send
```

Object classes

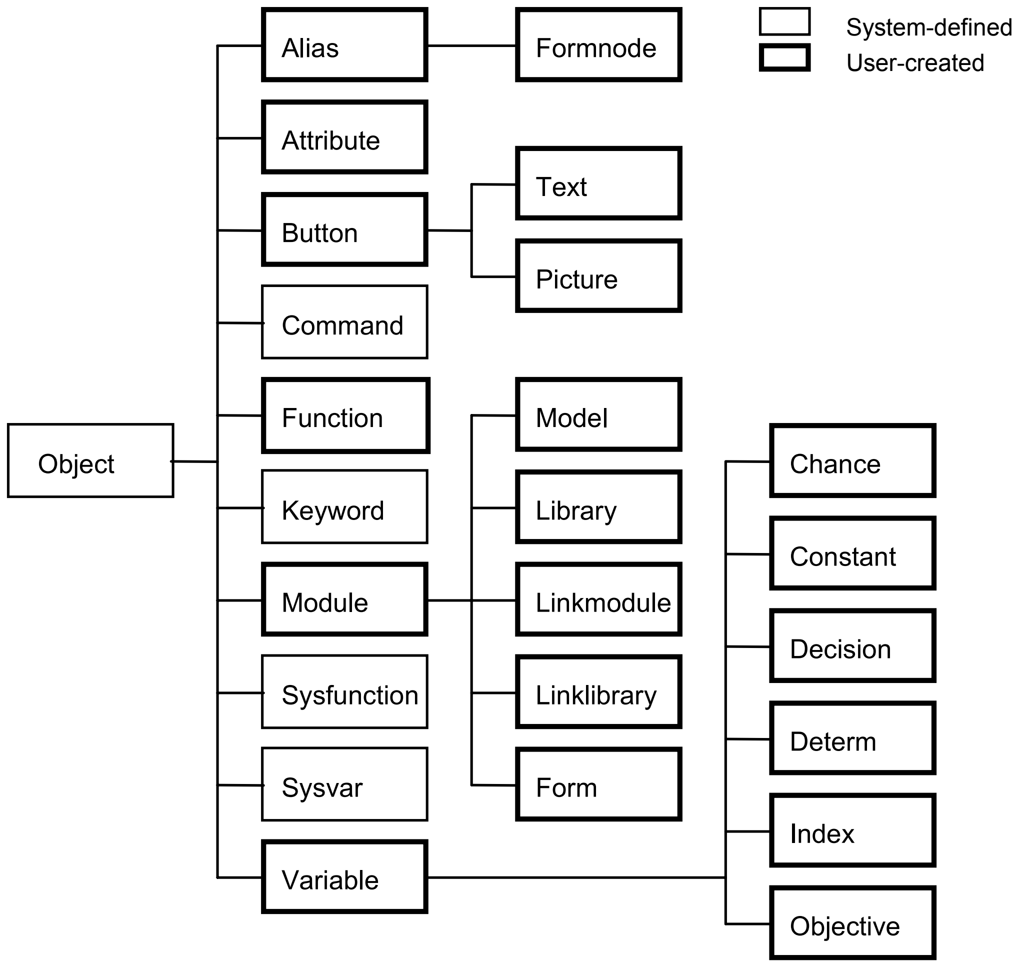
There are many different Object classes in Analytica, providing important elements of the modeling language. The following are the primary Object classes:

Alias	An object that represents a variable in a model other than the model containing the variable.
Attribute	A property of an object.
Command	An instruction from you to Analytica.
Function	A user-defined mathematical function that computes a value from its parameters**.**
Keyword	Words used in certain commands and expressions. For example, keywords are used in logical operators and in user-defined functions.
Module	An object that contains a set of variables and other user-defined objects.
Sysfunction	A standard built-in function, such as Sine (Sin), Standard Deviation (SDeviation), etc.
Sysvar	A pre-defined variable that controls the way Analytica formats information or executes certain commands.
Variable	An element of a model that can have a value.

Of the Object classes listed above, you can create variables, models, functions, aliases, and attributes. In addition, you can create more specific variable classes: Chance, Decision, Objective, Index, and Determ. You can create more specific model classes: Model, Version, Library, Linkmodule, and Linklibrary. Objects of the remaining Object classes are created by Analytica.

Class Hierarchy

Analytica provides a hierarchy of object classes. Some object classes inherit functionality from other object classes (that is, they are similar, with a few important differences).



Identifiers of Objects

The entities that Analytica recognizes are known as objects. If you create an object, its **identifier** can be up to 20 characters long (any characters beyond the 20th will be ignored). The first character must be a letter. The rest may be letters, digits, underscores (“_”), or periods (“.”). Analytica treats upper and lower case letters as equivalent (e.g., it is case-insensitive). The following are all legal identifiers:

```
a,    Alpha1,    OOOOO,    B.B.C.,
X12345678901,    net_value
```

▲ **Note** You may not include any other special characters, including spaces, in an identifier.

You can type the command “List” for a list of all predefined identifiers. However, this is a very long list. See the User Guide Appendix for a list of all predefined identifiers.

Abbreviations of Identifiers

Analytica permits **abbreviation** of identifiers in many, but not all, cases. You can abbreviate the identifier of an object when you type it at the prompt (“>“). If your abbreviation is ambiguous, Analytica will report an error and will not execute the command. In such cases, you must type the full identifier of the object. For example:

```
Fishinapond>sh
Syntax error:

?The Identifier 'Sh' is ambiguous.  Choose one of:
Show          Showhier       Showkey
Showundef
```

You cannot abbreviate identifiers of variables or other objects when you put them in the Definition of another object.

User-Specified Attributes

Objects may be described by various attributes. Some of these attributes may be specified by the user. These are known as user-specified attributes. For example, when you create a variable, you usually supply four attributes for it: Title, Units, Description, and Definition. The user-specified attributes are summarized below:

Author	The author(s) of a model or another object. By default, this is the computer's registered user name (on Macintosh, specified in the Sharing Setup control panel).
Title	Brief text used to identify a variable. Titles can be up to 255 characters; however Analytica often truncates them to fit.
Description	One or more lines about what an object represents. Descriptions will help you—and other people who access the model—understand it.
Definition	An expression that Analytica uses to compute the value of an object. A definition may be a simple number, an array, a probability distribution or a mathematical Expression that includes other variables. Definitions may be several lines long.
Units	The units of measurement of a variable, e.g., $/yr. You can leave the Units field blank if the variable is a number or a dimensionless ratio.
Parameters	When you create a new mathematical Function, you may specify a list of parameters for each Function. User-defined functions are described in the Analytica User Guide.

Check	A test that Analytica conducts on the validity of the value of a variable. A Check attribute is an Expression that contains the variable to be tested. If checking evaluates to False (or 0), Analytica warns you that the value of the variable is questionable. Check attributes are described later in this manual.
Usecheck	An attribute that controls whether a variable will (or won't) be checked for legality according to its Check attribute whenever it is evaluated. Usecheck overrides the system variable Checking for the particular variable to which it is attached.
Script	An attribute that contains a sequence of commands each separated by carriage returns that are executed when the user clicks on the button containing the script.
Balloonhelp	An attribute that contains help that appears when the mouse is over the object's node in a Diagram window. By default, Analytica uses the Description attribute for an object in Balloon Help. This attribute is available only on the Macintosh.
Indexvals	An attribute that may be defined as a list or list of labels, and function as a Self index for a variable table definition. This attribute is normally set by Analytica, but can be user-specified.
Domain	An attribute that may be defined as a list or list of labels, and function as a Self domain for a variable probtable definition. This attribute is normally set by Analytica, but can be user-specified.

Computed Attributes

In addition to user-specified attributes, there are *computed attributes* that Analytica adds to objects. Analytica generates the values of computed attributes. Other computed attributes used internally by Analytica are listed in the Appendix.

Date	The data the model was created.
Savedate	The last time the model was saved.
Saveauthor	The user who last modified a model, if different from the owner of the computer on which the model is created.
Inputs	A list of variables and functions that appear in the definition of the specified variable or function.
Outputs	A list of variables and functions that refer to this object in their Definitions.
Value	The deterministic value of a variable.
Probvalue	The probabilistic value of a variable.
Contains	The list of variables and other objects created within a model.
Isin	The model to which an object belongs.

User Interface Attributes

Many computed attributes are set by the graphical user interface in Analytica, and are called User Interface attributes. User Interface attributes are only represented visually and are usually not examined or modified directly by the user. These attributes range from window state information, to node location, color and font.

An object's node location and node size can be specified using the following attributes:

Nodelocation *h, v*	Horizontal and vertical coordinates (in Diagram space) of the node representing this object.
Nodesize *w, h*	Half-width and half-height (in Diagram space) of the node representing this object.

An object's information can be viewed in a number of windows; hence there are several different attributes used to store window state information, depending on the window type needed. The format of these window state attributes is as follows:

***attribute** version, left, top, width, height [, attribs]*

- *version* indicates the version of this attribute's format; 1 in the current version of Analytica
- *left* and *top* are the top-left screen coordinates of the window
- *width* and *height* are the window's width and height

Additional information regarding node style, diagram style, and result state may follow these values, depending on the attribute.

Nodeinfo *version, showinputs, showoutputs, showlabel, showborder, fill, usenodefont, formwidth, showbevel, showformicon*

Controls display options for a node, including arrows, node border, label and so on.

- *version* indicates the version of this attribute's format; 1 in the current version of Analytica

- *showinputs, showoutputs, showlabel, showborder, fill, usenodefont, showbevel* and *showformicon* have values of 0 (for off) or 1 (for on). *Showformicon* only applies to output nodes.
- *formwidth* specifies the width of an input node's or output node's field (0 is the default, which means the actual width will be computed based on the font size).

DiagramColor *red, green, blue*

Nodecolor *red, green, blue*

These are Diagram and Node background color attributes, respectively; light gray by default for diagrams, white by default for nodes. Color information is stored as RGB (Red, Green, Blue) components. Each RGB component is an integer value from 0 to 65,535. RGB color is additive, which means that as the value of a component is increased, the amount of that component in the total color decreases. An RGB color is black if all three components are set to 0; white if each component is set to 65,535. A negative value (-32767 to -1) is internally transformed by adding 65536.

Fontstyle *fontname, fontsize*

Nodefont *fontname, fontsize*

These are Diagram and Node font attributes, respectively, containing font name and font size. , *fontsize* is in pixels, not points. These are the same on a 1/72 dots per inch screen. VGA screens are usually 1/96 dots per inch, so an n pixel font size is 72/96 * n points.

Defaultsize *width, height*

This is the default half-height and half-width for a node, used when creating new nodes in a diagram.

Icon *hexdata* — Node icon information is stored as hexadecimal ASCII text. Analytica automatically "compiles" this information into the icon format when an icon is represented on the screen.

Graphsetup *commands*

Graphsetup contains a sequence of graph system variable assignments separated by carriage returns (and using the "~" continuation character), to be used when creating a graph result of the variable to which it is assigned.

Reformdef [*col, row***]**

Reformval [*col, row***]** — These are row/column reform state attributes for Edit Table and Result windows, respectively. *col* and *row*, inside brackets specify the index variable for the column and row, respectively.

Numberformat *version, formatcode, digits, zeroes, separators, currency*

Numberformat specifies how a number (result or output node) should be displayed.

- *version* is the current format version (1.0)
- *formatcode* has the following values:

code	**name**	**example**
D	Suffix	1.235K
E	Expoential	1.235e+4
F	Fixed point	12345.68
I	Integer	12346
%	Percent	1234567.80%
DD	Date	Tue, Oct 19, 1937
DB	Boolean	True

- *digits* specifies the number of digits for suffix and exponent formats.

- *zeroes* specifies the number of digits after the decimal point for fixed point, integer and percent formats.
- *separators* and *currency* only apply to fixed point formats; values are 0 (don’t show) or 1 (show).

Xyexpr *expr*	Specifies an expression to graph against, as an x-y scatter graph. This is used by the Analytica interface.

Inspecting Objects

To see all of the user-specified attributes of an object, you use the **Show** Command, as illustrated in the transcript below:

```
Example>show mpy
variable Mpy
Title: miles per year
Units: Miles/year
Description: Annual mileage driven
Definition: 12K
Nodelocation: 208,48
Nodesize: 52,24
```

To see the entire contents of a model (i.e., all of the user-defined objects in the model), you use the **Allshow** Command. Note that the output from Allshow may be as large as the model file itself.

To see a particular attribute of an object, you type the attribute and the object's identifier, as illustrated below:

```
Example>title mpy
miles per year
Example>
```

To see a summary of a variable, including its title, units, and value, you type its identifier. For example, to see a summary of variable Mpy (the entire variable is shown in the preceding example),

```
Example>mpy
mpy     Miles per year     (miles/year)  =
12K
Example>
```

To display the value of a variable, you ask for the Value attribute, as illustrated below:

```
Example>value mpy
12K
Example>
```

A discussion of possible values for variables is given elsewhere in this manual.

Current Objects

A *Current object* is the last object to be brought to Analytica's attention. If you type a command without specifying a parameter, Analytica will assume by default that you are referring to the current object, as illustrated below:

```
Example>show Mpy   (You do a "show" on variable
Mpy.)
variable Mpy
Title: miles per year
Units: Miles/year
Description:  Annual mileage driven
Definition: 12K
Nodelocation: 208,48
Nodesize: 52,24
Example>title      (You ask to see the Title
                of the current object.)
miles per year (This is the Title of Mpy.)
Example>
```

Modifying Attributes

You may add an attribute to an object, or replace an already assigned attribute, by typing the attribute to be added or replaced, the identifier of the object, a colon (":") and a new value or text string. For example, to change the title of variable Mpy:

```
Example>title mpy: mileage(You replace the Title.)
Example>title      (You ask to see the new Title
for Mpy.)
mileage
Example>title:miles per year  (Original Title
restored.)
Example>title
miles per year
Example>
```

You can add or replace only user-specified attributes, not computed attributes. To add or replace a definition of a variable (or another object), you type the identifier of the variable, a ":", and a new definition (you do not need to type the word "definition"). For example:

```
Example>mpy: 20K
```

is equivalent to:

```
Example>Definition mpy: 20K
```

If you include the identifiers of new objects in the definition of another object, Analytica will suggest creating them.

Deleting Objects

The **Delete** command allows you to remove variables from a model. **You cannot undo a Delete command.** Analytica will automatically update outputs of the object to be deleted so that they no longer reference the object.

List Command

To see a list of all objects that belong to a particular class, you use the **List** command. For instance:

```
Example>list sysfunction
Abs         Arctan        Array         Average
Beta
Chance      Choose        Cinterval     Concat
Correlation
Cos         Cumulate      Cumulative    Density
Dydx
Dynamic     Elasticity    Exp           Fixarray
Fractiles
Getfract    If0           Ifpos         Infromrec
Linear
Ln          Lognormal     Makerect      Max
Maxev
Mean        Mid           Min           Normal
Pred
Product     Rank          Rankcorrel    Reform
Repeat
Round       Sample        Sdeviation    Sequence
Sin
Size        Slice         Sqr           Sqrt
Subscript
Sum         Table         Uncumulate    Uniform
Vvariance
```

Objects and Attributes: Summary

Commands and other specific object identifiers are in **bold.** Generic objects, such as Class *class* or object *o*, are in *italic*:

Show *o*	To list all of the user-specified attributes of an object.
Allshow	To display all of the user-specified objects in a model.
class new-identifier	To create a new user-specified object (i.e., a variable, model, attribute, or function).
o	To list a summary of an object, you type its identifier.
attribute o	To display a particular attribute of an object, you type the attribute and the identifier of the object to be displayed. If you do not type an object after the attribute, Analytica will assume the current object.
attribute o:text	To change an attribute of a particular object. If you do not type an object after the attribute, Analytica will assume the current object.
var:x	To change the definition of a variable, you type its identifier, a colon, and a new definition.
Delete *o*	To delete an object.
List *class*	To see all of the objects of a particular class.
Profile *o*	Command to display all the attributes of an object *o* (both user-specified and computed). Primarily of interest to system developers.

Concepts:

object	An entity that Analytica recognizes.

User-Created object	An object that you create (e.g., a variable, model, attribute, function, or version).
System object	An object that Analytica supplies (e.g., a command, sysfunction, keyword, sysvariable, or kind). The user cannot create system objects.
attribute	A property of an object. For instance, variables have four attributes: a Title, Units, Description, and Definition.
Computed attribute	An attribute that Analytica computes automatically (i.e., Date, Modauthor, Inputs, outputs, Value, Probvalue, or Contains).
User Interface attribute	An attribute that Analytica computes using visual or other information in the user interface (i.e., Location, Nodesize, Windstate, NodeColor etc.).
Class	A category of object, such as command or variable.

3. Files and Editing

Module Files

Module files contain Analytica modules—they are collections of variables, functions, and other objects. There are two ways that you can create a module file: you can create variables and save them from within Analytica, or you can create module files outside of Analytica with a conventional text editor.

Format of 'Regular' Module Files

If you open a model using a word processor or text editor, or print the file, this is what it will look like:

```
{ From user korsan, Model Foxes_hares at Thu, Mar 23, 1995
       4:25 PM}
Softwareversion 1.0


{ System Variables with non-default values: }
Typechecking := 1
Checking := 1
Saveoptions := 2
Savevalues := 0


{ Non-default Time SysVar value: }
Time := Sequence(1,20,1)

Attribute Reference

Model Foxes_hares
Title: Foxes and Hares
Description: A simple ecology, showing the population
       changes over time when the fox population is
       dependent on a single species' (hare) population.
Author: Lumina Decision Systems
Date: Thu, Mar 2, 1995 10:15 AM
Saveauthor: korsan
Savedate: Thu, Mar 23, 1995 4:25 PM
Defaultsize: 44,20
Diagstate: 1,40,50,478,327,17
Diagramcolor: 32348,-9825,-8739
Fileinfo: 0,-1,4162,Model Foxes_hares,Foxes and Hares
Getresource Pagesetup,1

Variable Hare_birth_rate
Title: Hare birth rate
Description: Birth rate of the Hare population.
{etc.}
```

If you change any default settings during a session (see the chapter on System variables for more information about most defaults and modifications to them), Analytica will add these to the module file:

```
{ System Variables with non-default values: }
Typechecking := 1
Checking := 1
Saveoptions := 2
Savevalues := 0


{ Non-default Time SysVar value: }
Time := Sequence(1,20,1)
```

When you restart a model from the executive level of the computer, you will return to the point at which you ended your last work session with one exception—Analytica will have to perform all computations again from scratch.

Editing

You can create models, add objects to models, and create sets of numerical data outside of Analytica with an **editor** such as Microsoft® Word. Model Files created by editing can also contain Commands to Analytica. When the model File is started up, Analytica treats all objects and Commands in the text file as if they were entered interactively during a work session.

The transcript below illustrates how to create a model with the Microsoft Word application. To use another editor, you must be in the Finder, and not running Analytica.

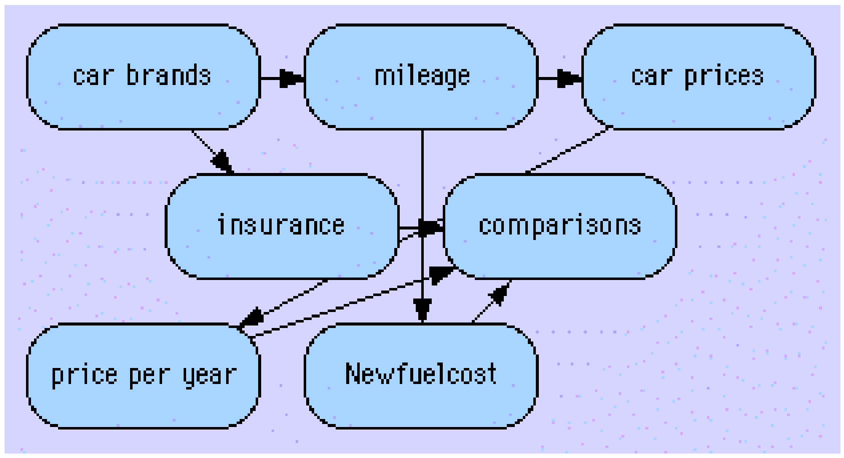
▲ **Note** When you create a model file in another editor, be sure that the names of the new objects **do not** overlap with names of objects in already-existing models. The only exception to this rule is when you are using the Update Command, discussed later in this chapter.

Models created in another editor must resemble interactive input. The objects in the file must have unique names. If you use a name that is already in use, Analytica will issue an error message when it tries to read the new model. Suppose you have started up Microsoft Word and typed in the following:

```
model Misc
```

```
Description: to demonstrate creating models in an
editor
variable CarType
Title: car brands
Units: names
Description:  Three popular car companies
Definition: ['vw', 'honda', 'bmw']
variable Newmpg
Title: mileage
Units: mi./gal.
Description: mileage, corresponding to brands in
CarType
Definition: array(cartype, [30, 35, 40])
variable Price2
Title: car prices
Units: $
Description: average car prices, corresponding to
brands in CarType
Definition: array(cartype, [5K, 10K, 15K])
variable Ins
Title: insurance
Units: $/yr
Description:  insurance rates for each brand in
CarType
Definition: array(cartype, [.5k, 1k, 1.5k])
variable Car_comp
Title: comparisons
Description: a table showing the various factors
related to CarType
Definition: [ppy, newfuelcost, ins]
variable Ppy
Title: price per year
Description: average amount paid per year for a
car (life span of 8 yrs.)
Definition: price2/8
variable Newfuelcost
Units: $/yr.
Description: The amount for fuel, based on 15K
mi/yr, $1.20 per gallon.
Definition: (15K*1.2)/newmpg
Close Misc All model Files must end
    with a "close" statement.
```

Start up file Misc.mod in Analytica.



If the nodes do not have Nodelocation attributes, Analytica will give nodes non-overlapping default locations.

Adding Modules

There may be circumstances under which you may want to add a linkmodule or linklibrary not already contained in the model. Linkmodules and linklibraries can be added to a model using the Include command. The parameters to the Include command resemble the contents of the Fileinfo attribute (See Objects and their Attributes, File Info).

Include *alias, class object, unused, platformID, pathID,pathname*

Read in a module or library from the specified file's location on a disk.

- *alias* refers to a Macintosh File alias (type 'alis') resource ID, or 0 if none is specified. If a non-zero value is specified and the resource exists, Analytica uses the Alias manager to resolve the location of the file and ignores the additional information in this attribute. If a zero value is specified or if the resource doesn't exist, Analytica uses the additional information stored in this attribute to locate the file. For Windows, this value should always be 0.
- *class* and *object* are this object's class and identifier, respectively, used to check consistency.
- *unused* is reserved for future use.

- *platformID* represents the platform the fileinfo is saved on, and may have the following values:

value	**platform**
1	Macintosh (HFS)
2	Windows

  • *pathID* is used by the MacOS at runtime to identify a working directory for the file.

- *pathname* is a path name which is relative to the including file.

▲ **Note** When you add a Linkmodule or Linklibrary to a model, it doesn't actually become a part of that model; only a link is created. If the model is saved with this file reference information, it will automatically embed Include commands so that the next time the model starts up it automatically includes the specified linkmodules.

To transform a Linkmodule or Linklibrary into a non-filed module or library, change it class after reading it in and before saving any changes.

Updating an Existing Model

There are two ways to use a text editor to **update** an already-existing model. First, you can edit a model that is in your directory. Or, you can create a new File that contains objects to be included a model at some later time. If you choose to create a separate File, you must begin the File with the keyword, **update** and end it with **end update**. You can then start up another model and *Read* in the Update File, as illustrated below:

```
(An editor is opened.  The user types the
following:)
update
model Test
variable Price
Title: car prices
Units: $
Description: average car prices, corresponding to
brands in CarType
Definition: array(cartype, [10k, 15k, 20k])
close
```

```
end update
(Save the model and quit the editor)
(Start up model Misc. in Analytica)
Misc> show price
variable Price
Title: Car prices
Units: $
Description: average price paid for a car.
Definition:  10K
Misc>readfile test (The user asks Analytica to
    "read in" model Test.)
Reading from test
#
the definitions of all variables and functions are
OK.
Misc>show price
variable Price  Analytica refers to the
Title: car prices  Version of Price
Units: $   that was defined in model Test.
Description: average car prices, corresponding to
brands in CarType
Definition:  array(cartype, [10k, 15k, 20k])
Misc>
```

Data Files

If you create a data file outside of Analytica, you must format the file to resemble Analytica input—for example, sets of numerical data must be formatted as variables. You should also begin the file with the term **update** and end it with **end update**. By using the Update Command, you can read this file into any other model. As noted above, the Update Command tells Analytica to look for object names that are shared between the already-existing model file and the File to be read. If Analytica finds overlapping objects, it will replace any attributes in the already-existing model with the new ones from the file that is being read. As noted above, the Read Command causes Analytica to treat an input file in the same way it treats "live" input from the terminal.

Models and Editing: Summary

Commands:

Readfile *model*	To "load in" a model File while working in Analytica, adding it to the current project.

Include *alias, class object, unused, platformID, pathID,pathname*

Read in a module or library from the specified file's location on a disk.

- *alias* refers to a Macintosh File alias (type 'alis') resource ID, or 0 if none is specified. If a non-zero value is specified and the resource exists, Analytica uses the Alias manager to resolve the location of the file and ignores the additional information in this attribute. If a zero value is specified or if the resource doesn't exist, Analytica uses the additional information stored in this attribute to locate the file. On Windows, alias should always be 0.
- *class* and *object* are this object's class and identifier, respectively, used to check consistency.
- *unused* is reserved for future use.
- *platformID* represents the platform the fileinfo is saved on, and may have the following values:

value	**platform**
1	Macintosh (HFS)
2	Windows

- *pathID* is used by the MacOS at runtime to identify a working directory for the file. Under Windows, this value is always 0.
- *pathname* is a path name which is relative to the including file.

Concepts:

Close	Statement at end of model when updating it in a text editor.
Readfile	Incorporates a model File in the current model.

Update	Statement that begins a model File that is updated in a text editor.
End Update	Statement that ends a model File being updated in a text editor.

4. Arrays

This chapter describes how to access arrays and control the format of arrays for output. Before reading this chapter, you should already be familiar with arrays as described in the Analytica Reference.

Arrays are created from a variety of sources. Two functions for creating arrays include Table and Array. Also, variables and Index variables defined as Lists (e.g., for parametric analysis) are arrays.

The Table Function

In parametric analysis, two and three-dimensional Arrays can be generated from expressions containing one, two, or three Index variables. The **Table** Function is designed to permit you to input a Table directly and specify its Indices within one variable. The parameters to Table must be enclosed in parentheses, as shown below:

```
Example>Mpg:Table(Cartype) (32, 34, 18)
Example>value mpg
   Cartype        vw,   honda,     bmw
           [      32,      34,      18]
```

You may specify more than one Index variable in a Table. The number of Index variables specifies the number of Dimensions. The number of values in the Table (specified in the second list of parameters) must equal the product of the numbers of elements for every dimension. For instance, if the first parameter of the Table is two Indices of three elements each, the second parameter must have nine elements.

```
Example>variable yr
Title:years
Units:ad
Description:model year of car
Definition:[1985, 1986, 1987, 1988]
Example>variable car_prices
Title:prices for cars
Units:$
Description:Prices for three brands of cars over a
three-year period.
```

```
Definition:table(cartype, yr)(8K, 9K, 9.5K, 10K
12K, 13K, 14K, 14.5K 18K, 20K, 21K, 22K)
Example>value car-prices
   Yr              1985,    1986,    1987,
1988
   Cartype
            vw [    8000,    9000,    9500,
10K]
         honda [     12K,     13K,     14K,
14.500K]
           bmw [     18K,     20K,     21K,
22K]
```

The Array Function

The **Array** Function is similar to the Table Function, in that it can be used to specify an Array directly, but its syntax is a bit different. Like Table, the first *n* parameters are the Index variables that specify the n dimensions of the result. The main difference is that the values to be the elements of the Array are listed in square brackets as the last parameter (instead of as an extra list between parentheses as in Table). For example:

```
Example>Mpg: Array(Cartype, [32, 34, 18])
Example>Value
   Cartype      VW,  Honda,    BMW
          [     32,     34,     18]
```

If the Array has multiple dimensions, then the elements are listed in nested brackets, following the structure of the Array as a list of lists (of lists ... etc., according to the number of dimensions):

```
Example>variable Car_prices
Title: Prices for car
Units: $
Description: Prices for three brands of car over
four year period.
Definition: Array(Cartype, Yr, [[8K, 9K,  9.5K,10K
], [12K, 13K, 14K, 14.5K],[18K, 20K, 21K, 22K  ]])
```

As in Table, the Index variables are specified from the outer to innermost. In the above example Cartype comes before Yr, and so the Array is specified as a list (indexed by CarType) of lists (each indexed by Yr). The result looks the same as before:

```
Example>value car_prices
   Yr              1985,   1986,   1987,   1988
   Cartype
            VW [   8000,   9000,   9500,    10K]
         Honda [    12K,    13K,    14K,14.500K]
           BMW [    18K,    20K,    21K,    22K]
```

Note that the size of each list in square brackets must match the size of the corresponding Index. In this case there is a list of three elements (for the three car types), and each element is a list of four elements (for the four years). Analytica will complain if these sizes don't match. Size is discussed below.

Size of a Dimension

The function **Size** returns the number of elements of a list:

```
Example>Size([10,20,30])
3
```

If it is applied to a multi-dimensional array, it returns the number of elements in all dimensions:

```
Example>size([[1,2,3], [4,5,6]])
6
```

Selecting Parts of an Array

If you want to extract a single element of an array, or a column or a row of a table, you can specify the index and its value in square brackets after the array. Suppose we reassign a single value to Price in the Fuel Cost model, so that Cost is again 2 dimensional:

```
Example>Gasprice:1.05
Example>value FuelCost
   Mpg              24,     28,     32,     36
   Distance
           200 [8.7500, 7.5000, 6.5625, 5.8333]
           250 [10.937, 9.3750, 8.2031, 7.2916]
           300 [13.125, 11.250, 9.8438, 8.7500]
```

You can extract a single row or column from this table by specifying the index and its value in square brackets, after the variable name:

```
Example>FuelCost[Distance=250]
   Mpg        24,     28,     32,     36
         [10.937, 9.3750, 8.2031, 7.2916]
Example>Cost[Mpg=36]
   Distan    200,    250,    300
         [5.8333, 7.2916, 8.7500]
```

You can also select over more than one index dimension at a time. The index value specifications are separated by comma(s).

```
Example>FuelCost[Distance=250, Mpg=36]
7.2916
```

Note that in most computer languages you need to know which index is attached to which dimension of the array, and you have to specify them in the right order, row index before column index or vice versa. In Analytica you don't need to remember this, and instead specify the indexes by identifier in whatever order you like.

The Slice Function

The **Slice** Function returns a cross-section of an array. The syntax is:

Slice(*a, i, n*) **-** Analytica returns the part of array *a* that corresponds to index variable *i* and is restricted to the *nth* elements of *i*, as shown below:

```
Example>sho cartype
variable Cartype
Title: car brands
Units: names
Definition: ['vw', 'honda', 'bmw']
Example>sho price
variable Price
Title:  Car prices for each of three types of
cars.
Units:  $
Description:  Total purchase price for three cars.
Definition:  array(cartype, [5K, 10K, 15K])
Example>mid cost   (Cost is an Output of Price.)
   Cartype              vw,     honda,       bmw
   Mpg
            15 [      2185,      2810,      3435]
```

```
           30 [      1705,      2330,      2955]
           40 [      1585,      2210,      2835]
Example>slice(cost, cartype, 1)
   Mpg               15,        30,        40
             [      2185,      1705,      1585]
```

In the example above, Analytica returns the values corresponding to the first element of Cartype (i.e., the values of VW). **Note:** the third parameter of Slice can be an array, too.

```
Example>slice(cost, cartype, [1, 2])
   Mpg                 15,        30,        40
            1 [      2185,      1705,      1585]
            2 [      2810,      2330,      2210]
```

The Subscript Function

Subscript*(a, i, x)* - This is very similar to Slice; however, instead of referring to the *position* of the element(s) in *i*, Subscript refers to the actual value, *x* within *i*.

```
Example>mid cost   (Cost is an Output of Price.)
   Cartype              vw,     honda,       bmw
   Mpg
           15 [      2185,      2810,      3435]
           30 [      1705,      2330,      2955]
           40 [      1585,      2210,      2835]
Example>subscript(cost, mpg, 15)
   Cartype            vw,     honda,       bmw
             [      2185,      2810,      3435]
Example>subscript(cost, cartype, ['vw', 'honda'])
   Mpg                 15,        30,        40
            1 [      2185,      1705,      1585]
            2 [      2810,      2330,      2210]
```

Subscript permits you to use arbitrary expressions as the first parameter. For instance:

```
Example>mpg:[32 34 18]
Example>subscript(cost/12, mpg, 32)
   Price            10K,      15K,      25K
   Cartype
           vw [ 193.229, 245.313, 349.479]
        honda [ 176.302, 217.969, 301.302]
          bmw [ 156.399, 186.161, 245.685]
```

If you tried to type this expression without using the Subscript Function, Analytica would have printed an error message:

```
Example>cost/12[mpg=32]
? Expecting "end of line".
cost/12[mpg=32]
        ^?  Syntax error:
```

Re-formatting Arrays

Normally, Analytica automatically controls the row and column choices that determine the order of dimensions, or format, in which a table will be displayed to the user. This information is stored in User Interface attributes *Reformdef* and *Reformval*. However, there may be occasions when you wish to control the format of array, for example via IAC. There are two main methods for reformatting arrays: using the Reform function, and controlling tabletype and delimiter system variables.

The Reform Function

The **Reform** function changes the displayed structure of an array. Reform is useful in altering tables to look the way you want them to look.

Reform(*x*, [*i1*, *i2*. . .*in*]) prints out an *n*-dimensional array with index i n along rows, i [n-1] down columns, and the rest of the indices (if any) choosing the sequence of tables (2-D slices) from the array, so that the earliest index varies most slowly. This is to allow you to print out 2 or more dimensional arrays in whatever sequence you like most. The first argument must be an array with 2 or more dimensions. The second argument is a list of Indices, which must be indices of *x*, but may be in any order.

```
Example>mpy:[8k 12k 15k]
Example>mpg:[26 30 35]
Example>gasprice:1
Example>fuelcost:gasprice*mpy/mpg
Example>mid fuelcost
   Mpg                26,      30,      35
   Mpy
          8000 [ 307.692, 266.667, 228.571]
           12K [ 461.538,     400, 342.857]
            15 [ 0.57692, 0.50000, 0.42857]
```

```
Example>reform(fuelcost, [mpg, mpy])
   Mpy              8000,     12K,      15
   Mpg
            26 [ 307.692, 461.538, 0.57692]
            30 [ 266.667,     400, 0.50000]
            35 [ 228.571, 342.857, 0.42857]
```

Controlling Tabletype and Delimiter

A variable value result that is an array of two or more dimensions is usually shown in a readable format designed for Typescript viewing. This format is not ideal when the result is intended for export to another spreadsheet program, for copying and pasting within Analytica, or for IAC. Two system variables in Analytica, Tabletype and Delimiter, make it easier to read values of variables back into Analytica, into other spreadsheet-oriented programs, or for IAC.

A system variable called Tabletype controls the formatting and content for the display of values in the typescript, regardless of whether they are deterministic or probabilistic, a simple array, or a higher-dimensional table.

Setting Tabletype to 0 displays tables in the format that is readily understandable when viewed in the typescript (default). Setting Tabletype to 1 displays tables in a format that can be used as a Analytica definition. Setting Tabletype to 2 prints out tables in a format that can be used to import data into a spreadsheet or other data analysis program, or for IAC.

For example, with the default Tabletype of 0, tables are printed in the usual manner.

```
Carcost>value Fuelcost
   Mpg               25,     30,     35
   Mpy
           12K [    576,    480,411.429]
           18K [    864,    720,617.143]
           24K [   1152,    960,822.857]
```

When we set the Tabletype to 1, tables are printed in a format that Analytica can accept as a valid definition.

```
Carcost>tabletype: 1

Carcost>value Fuelcost
Table(Mpy,Mpg)(    { Notice that the values of
       576,    480,411.429,    indices are not
displayed }
       864,    720,617.143,
      1152,    960,822.857
)
```

You can take such a table, copy it using the Copy menu command in the Edit menu (C), and paste it into the input portion of the Analytica typescript, i.e., click in the input portion and use the Paste menu command in the Edit menu (V). You can do the same to copy and paste a definition into an Analytica model text file.

With Tabletype set to 2, tables are printed in a format that you can use to copy and paste into a spreadsheet program or another data analysis program.

```
Carcost>tabletype: 2

Carcost>value Fuelcost

Mpg         25  30  35
Mpy
    12K     576 480 411.429
    18K     864 720 617.143
    12K     1152    960 822.857
```

This is a tab-delimited table of five rows and four columns, with the second row containing only one column. Follow the same Copy and Paste commands described above to copy a table from Analytica into a spreadsheet or other data analysis package. This tab-delimited format is also ideal for IAC.

If you wish to export the value of an Analytica variable into another program such as a spreadsheet program, a graphing package, or a statistical package, you need to have the data in a format that the other program can read in easily. Most of these programs recognize text files containing rows and columns of text and numbers separated by special characters called delimiters. Tabletype=2 prints tables in a similar format to the Tabletype=0 format, except it doesn't presume commas for its delimiter and doesn't print out brackets surrounding the one-dimensional parts of the table. Instead, Tabletype=2 uses another system variable called Delimiter which separates items in the table.

Delimiter=0 uses commas in tables which some programs require for recognizing input. Delimiter=1 uses tab characters which can't be seen but are printed in the Analytica typescript and are the most common delimiter type (default and shown in the example in this section). Delimiter=2 uses space characters which are another common delimiter type in table-oriented programs.

System variable Delimiter only applies to Tabletype=2 since both the typescript-formatted and Analytica-formatted tables rely on using comma delimiters.

▲ **Note** In order to create true x-y graphs (in some programs, such as graphing programs and spreadsheets) you must provide an additional x column for numerical data following the first column. For these programs, you can use Tabletype: 3. The output is identical to Tabletype: 2, except the first column is repeated twice when the column contains numbers.

Arrays: Summary

Array(*i1, i2*, ... in, y)	A function, which assigns a list of indices, *i1, i2, ... in,* as the indices of the array *y,* with *i1* as the index of the outermost dimension, *i2* as the second outermost, etc. *y* must have at least *n* dimensions.
Extraction	You can “pull out” specific values from an array by requesting a particular index and optionally, particular values in the index (e.g., **cost[mpg=15]**).
Slice(*a, i, n*)	Function that returns the *n*th value of array *a* over the dimension indexed by *i*. *n* must be between 1 and the length of *i*. *n* may also be an array of values, in which case, Analytica will return an array of corresponding values from *a*.
Subscript(*a, i, x*)	Function which gives the element of array *a* for which index *i* has value *x*. *x* must be one of the values of index *i*. *x* may also be an array of values from index *i*, in which case it will produce a corresponding array of resulting values from *a*. (It is essentially the same as *a*[*i*=*x*], but allows *a* to be a general expression, instead of restricting it to a variable.)
Size(*x*)	A function that returns the number of elements of the outermost dimensions of an array *x*.
Table(*i1, i2, ... in*) (*x1, x2, x3, ... xm*)	This function creates an *n*-dimensional array, indexed by the Indices *i1, i2, ... in.* The number of indices may be 1 or more. The indices must be separated by commas and enclosed in parentheses, as shown. The second set of parameters to Table specify the values that go into the array. These are also enclosed in parentheses, and the separating commas are optional. Each

	of these values is specified by an expression *x*. The number of values required is the number of elements of the array, which is the product of the sizes of all the dimensions. In this list of elements, the last index *in* is the innermost, varying most rapidly.
Reform(*x,* [*i1, i2. . .in*])	Command to print a multi-dimensional array *x* in a sequence so that index *i1* is varying slowest, *i2* next slowest and so on. The indices *i1, i2,* etc., must be all of the indices of *x*.

5. System Variables

System variables (abbreviated as **sysvars**) are pre-defined variables in Analytica. You can modify system variables to control the way things are printed, plotted etc. Many system variables control the formatting of graphs. You can also include system variables in the definitions of user-specified variables.

You can use the Show command to print a system variable. You may change the definition of a system variable with the usual syntax for definition assignment, e.g.,

```
Example>checking:0
```

If the definition is outside of the legal range for that variable, Analytica will issue the following message:

```
Example>checking:4
 Value for Checking must be an integer between 0
and 1 inclusive.
Do you want to edit the Definition of Checking?
```

System variables for controlling the format of graphs and other plots are listed in the Summary section of the chapter on graphing.

Abbreviation	System variable determining Analytica' ability to understand abbreviations. **abbr: 1** means abbreviations are accepted, **abbr: 0** means abbreviations are not permitted.
Checking	System variable telling Analytica to examine all of the Check attributes in a model and to flag the first incidence of a definition that is in conflict with its check. **checking:1** switches it on (default); **checking:0** switches it off
Heapsize	System variable, printed out when you use the Clock command. The amount of memory in use by Analytica.
Normal_fracs	System variable used in calculating normal distributions.

Numberwidth	A System variable that controls the width used in printing out numbers. Default is 4. Values less than 4 aren't recommended. If Numberwidth is set to 12 or more, numbers will be printed in floating point "E" format (e.g., 1.234456789E+12). The value 0 means "free format", and Analytica will choose what width it likes. *Syntax*: **numberw:** *x*
Run	System variable that indexes a sample distribution.
Samplesize	System variable that determines the number of elements in a sample distribution. **samplesize:100** is the default; **samplesize: 32000** the maximum.
Sampletype	System variable specifying the procedure for sampling a distribution, i.e., Median Latin Hypercube **(Sampletype: 0)**, Random Latin Hypercube **(Sampletype: 1)**, or Simple Monte Carlo **(Sampletype: 2)**
Time	System variable, usually used as an index for Dynamic. For dynamic models, this must be assigned a list of the time points (e.g. years) at which the Dynamic variables are to be evaluated.
Verbosity	System variable. Controls how “chatty” Analytica is. **verb: 1** verbose; **verb: 2** does not print tables unless explicitly requested; **verb: 4** prints "evaluating <var>" when it is; **verb: 8** is Filetrace; **verb: 16** is debug mode.

Appendix A: Language Summary

In the following descriptions, the name of each Analytica object is in **Bold** face, and Analytica concepts or terms without corresponding objects are in *italic*. Optional parameters are enclosed in *[...]*. In general, if you omit the main parameter of a command, Analytica will assume you mean the current object (i.e., the last object explicitly mentioned). If you omit the last parameter of a function such as Sum, Min, or Max, whose second parameter is an optional index specifying the dimension of a multi-dimensional array over which to perform the operation, Analytica will assume the outermost dimension.

Attributes

Attributes fall under the following categories:

- **User-specified attributes**—attributes that can be created and/or modified by the user. You assign a value *v* to an attribute *a* of object *o* with the following syntax: *a o*: *v*. In general, to display an attribute of a particular object you use the syntax *a o*.
- **Computed attributes** —attributes that are computed by Analytica.
- **User Interface attributes** —attributes that are set by visual manipulation in the Analytica user interface.

User-specified Attributes

Author	The author(s) of a model or another object. By default, this is the computer's registered user name (on Macintosh, specified in the Sharing Setup control panel).
Title	Brief text used to identify a variable. Titles can be up to 255 characters; however Analytica often truncates them to fit.
Description	One or more lines about what an object represents. Descriptions will help you—and other people who access the model—understand it.
Definition	An expression that Analytica uses to compute the value of an object. A definition may be a simple number, an array, a probability distribution or a mathematical Expression that includes other variables. Definitions may be several lines long.
Units	The units of measurement of a variable, e.g., $/yr. You can leave the Units field blank if the variable is a number or a dimensionless ratio.
Parameters	When you create a new mathematical Function, you may specify a list of parameters for each Function. User-defined functions are described in the Analytica Reference.
Check	A test that Analytica conducts on the validity of the value of a variable. A Check attribute is an Expression that contains the variable to be tested. If checking evaluates to False (or 0), Analytica warns you that the value of the variable is questionable. Check attributes are described later in this manual.

Usecheck	An attribute that controls whether a variable will (or won't) be checked for legality according to its Check attribute whenever it is evaluated. Usecheck overrides the system variable Checking for the particular variable to which it is attached.
Script	An attribute that contains a sequence of commands, each separated by carriage returns, that are executed when the user clicks on the button containing the script.
Balloonhelp	An attribute that contains help that appears when the mouse is over the object's node in a Diagram window. By default, Analytica uses the Description attribute for an object in Balloon Help. This attribute is only available on the Macintosh.
Indexvals	An attribute that may be defined as a list or list of labels, and function as a Self index for a variable table definition. This attribute is normally set by Analytica, but can be user-specified.
Domain	An attribute that may be defined as a list or list of labels, and function as a Self domain for a variable probtable definition. This attribute is normally set by Analytica, but can be user-specified.

Computed Attributes

Date	The data the model was created.
Savedate	The last time the model was saved.
Saveauthor	The user who last modified a model, if different from the owner of the computer on which the model is created.
Inputs	A list of variables and functions that appear in the definition of the specified variable or function.
Outputs	A list of variables and functions that refer to this object in their Definitions.
Value	The deterministic value of a variable.
Probvalue	The probabilistic value of a variable.
Contains	The list of variables and other objects created within a model.
Isin	The model to which an object belongs.

User Interface Attributes

Nodelocation *h, v*	Horizontal and vertical coordinates (in Diagram space) of the node representing this object.
Nodesize *w, h*	Half-width and half-height (in Diagram space) of the node representing this object.

Nodeinfo *version, showinputs, showoutputs, showlabel, showborder, fill, usenodefont, formwidth, showbevel, showformicon*

Controls display options for a node, including arrows, node border, label and so on.

- *version* indicates the version of this attribute's format; 1 in the current version of Analytica
- *showinputs, showoutputs, showlabel, showborder, fill, usenodefont, showbevel* and *showformicon* have values of 0 (for off) or 1 (for on). *Showformicon* only applies to output nodes.
- *formwidth* specifies the width of an input node's or output node's field (0 is the default, which means the actual width will be computed based on the font size).

DiagramColor *red, green, blue*

Nodecolor *red, green, blue*

These are Diagram and Node background color attributes, respectively, light gray by default for diagrams, white by default for nodes. Color information is stored as RGB (Red, Green, Blue) components. Each RGB component is an integer value from 0 to 65,535. RGB color is additive, which means that as the value of a component is increased the amount of that component in the total color decreases. An RGB color is black if all three components are set to 0; white if each component is set to 65,535. A negative value (-32767 to -1) is internally transformed by adding 65536.

Fontstyle *fontname, fontsize*

Nodefont *fontname, fontsize*

These are Diagram and Node font attributes, respectively, containing font name and font size. fontsize is in pixels, not points. These are the same on a 1/72 dots per inch screen. VGA screens are usually 1/96 dots per inch, so n pixels fontsize is 72/96 * n dots per inch.

Defaultsize *width, height*

This is the default half-height and half-width for a node, used when creating new nodes in a Diagram.

Icon *hexdata* Node icon information is stored as hexadecimal ASCII text. Analytica automatically “compiles” this information into the Icon format when an Icon is represented on the screen.

Graphsetup *commands*

Graphsetup contains a sequence of graph system variable assignments separated by carriage returns (and using the “~” continuation character), to be used when creating a graph result of the variable to which it is assigned.

Reformdef [*col, row***]**

Reformval [*col, row***]** These are row/column reform state attributes for Edit table and Result windows, respectively. *col* and *row*, inside brackets, specify the index variable for the column and row, respectively.

Numberformat *version, formatcode, digits, zeroes, separators, currency*

Numberformat specifies how a number (result or output node) should be displayed.

- *version* is the current format version (1.0)
- *formatcode* has the following values:

code	**name**	**example**
D	Suffix	1.235K
E	Expoential	1.235e+4
F	Fixed point	12345.68
I	Integer	12346
%	Percent	1234567.80%
DD	Date	Tue, Oct 19, 1937
DB	Boolean	True

- *digits* specifies the number of digits for suffix and exponent formats.
- *zeroes* specifies the number of digits after the decimal point for fixed point, integer and percent formats.
- *separators* and *currency* only apply to fixed point formats; values are 0 (don't show) or 1 (show).

Fileinfo *alias, class object, unused, platformID, pathID, pathname*

Information about a file's location on a disk.

- *alias* refers to a Macintosh File alias (type 'alis') resource ID, or 0 if none is specified. If a non-zero value is specified and the resource exists, Analytica uses the Alias manager to resolve the location of the file and ignores the additional information in this attribute. If a zero value is specified or if the resource doesn't exist, Analytica uses the additional information stored in this attribute to locate the file. On Windows, alias should always be 0.
- *class* and *object* are this object's class and identifier, respectively, used to check consistency.
- *unused* is reserved for future use.

- *platformID* represents the platform the fileinfo is saved on, and may have the following values:

value	**platform**
1	Macintosh (HFS)
2	Windows

- *pathID* is used by the MacOS at runtime to identify a working directory for the file.
- *pathname* is a path name which is relative to the including file.

Commands

There are several categories of commands:

- **Inspection commands** - Commands for displaying objects and attributes.
- **File commands** - Commands for creating various kinds of files.
- **User Interface commands** - Commands for manipulating the Analytica user interface.
- **Miscellaneous commands -** Commands for starting Analytica, ending a work session, getting help, and assorted other Commands.

In general, parameters to commands are optional. If no parameter is specified, Analytica will apply the command to the current object. If a parameter is required and none has been specified, Analytica will display an error message and will not execute the command.

Inspection Commands

Show *x*	To display an object's user-specified attributes**.** See also Show Diagram, etc. in User Interface Commands.
Allshow *m*	To do a Show on model *m* and all of the objects in it. If no model is specified, the current model is assumed**.**
What *o*	Displays a summary of an object *o*. If *o* is a variable it prints its identifier, title, units and value (or indexes of its value if it is an array).
Why *v*	Prints the identifier, Title, units, and value of variable *v* and of all those variables on which it depends (its Inputs). Useful for examining why the variable has the value it does.
Whyall	Shows the Inputs of all objects in a model**.**

List *k*	Command to display all of the objects of a particular kind *k*. If *k* is omitted, Analytica will display *all* objects, a horribly long list you probably won't want to see.
Tree *o*	Command to display the Inputs of an object (variable, Function, or model), the Inputs of those Inputs, and so on.
Profile *o*	Command to display all the attributes of an object *o* (both user-specified and computed). Primarily of interest to system developers.

File Commands

.i.**Include** *alias, volRefNum, dirID, class object, filename*

Read in a module or library from the specified file's location on a disk.

- *alias* refers to a Macintosh File alias (type 'alis') resource ID, or 0 if none is specified. Only a model file should specify a non-zero value. If a non-zero value is specified and the resource exists, Analytica uses the Alias manager to resolve the location of the file and ignores the additional information in this attribute. If a zero value is specified or if the resource doesn't exist, Analytica uses the additional information stored in this attribute to locate the file. On Windows, alias should always be 0.
- *volRefNum* and *dirID* are Macintosh HFS specifiers that help locate the file.
- *class* and *object* are this object's class and identifier, respectively, used to check consistency.
- *filename* is the actual file name, or pathname.

Update Command that puts Analytica into update mode, so that it allows redeclarations of already created

	objects. Existing attributes of such objects will be overwritten by the attributes of the new Versions. This mode is used when reading in new Versions of selected objects**.**
Endupdate	A Command used to finish an Update File. It changes the Analytica mode to make it illegal to attempt to redeclare existing objects**.**

User Interface Commands

Open Diagram *m*	Opens or brings to front the Diagram window for model m.
Open Object *o*	Opens or brings to front the object window for object o.
Open Result *v*	Opens the Result window for variable v, in the form it was last viewed.
Close Diagram *m*	Closes the Diagram window for model m.
Close Object *o*	Closes the object window for object o.
Close Result *v*	Closes the Result window for variable v.

Miscellaneous Commands

Bye	Command that ends a work session. Analytica will prompt for a File to store the current model.
Clock	To see the current date, time, and memory usage.
Close *m*	In a model File, Close specifies the end of the current model. When used as a Command interactively, Close moves you from a Submodel to its Parent.
News	To read news about Analytica.

Object Manipulation Commands

Delete *o*	A Command to remove an object *o* from a model. You cannot remove user-created objects that have other objects depending on them (outputs). If *o* has outputs, you must Delete or Rename the outputs.
Rename *o*	To assign a new identifier to an object. When you rename a variable that has outputs, each output variable's definition is automatically updated to replace the old identifier with the new identifier.
Edit *v*	Opens up the appropriate window that allows editing the definition for variable v.
Move o *m*	To move an object *o* from the model in which it currently resides to the specified model *m*.
Open *m*	To make a model *m* the current model.

Keywords

Keywords fall under the following categories:

- **Parameter Type Qualifiers -** specifications restricting the legal values of Function parameters. These are listed in the Analytica User Guide.
- **Parts of Expressions** - Keywords used for modifying Commands, logical operators, and parts of conditional statements.

Parts of Expressions

And	A logical operator. Returns "true" only if both its operands are true. *Syntax: x* **and** *y*.
Or	A logical disjunction operator returning true if either *x* or *y* or both are true. *Syntax: x* **or** *y*
Not *x*	A logical negation operator (if *x* is false (0) then true, else false).
If	Part of a conditional statement. *Syntax:* **if** *x* **then** *y* **else** *c* If *x* is true (not zero) Analytica returns the value of *y*, otherwise it returns the value of *z*. The **else** part is required. *x*, *y* and/or *z* may be arrays.
Then	Keyword used as part of a Conditional Statement. Then must be used with If and Else. *syntax:* **if** *x* **then** *y* **else** *z*.
Else	A keyword used in conditional statements. *Syntax:* **if** *x* **then** *y* **else** *z* .
Of	Separator for object attribute reference. *Syntax: attribute* of *object*.

Classes

Object classes are listed below. Other classes form a hierarchy, and generally fall under the following parent classes:

- Object
- Variable
- Model

Each specified object class inherits the behavior of its parent class. The following classes have a parent class of **Object**.

Attribute	An object class. attributes are properties of objects, such as Title, Description, Definition, or value. It is possible to create new attributes. Prompting for the new attribute is controlled with the Askattribute Command.
Object	A category of object, such as a variable, attribute, or class (!).
Command	An instruction from the user to Analytica.
Function	A user-defined mathematical function that computes a value from its parameters**.**
Keyword	A term used for various purposes, e.g., a Qualifier specifying non-default evaluation for a User-Defined Function or a logical or conditional operator.
Localvar	A local variable or parameter within a Function. Used internally by Analytica.
Module	A model contains a collection of variables and other objects.
Variable	A basic component of an Analytica model. variables can represent any kind of quantity. They generally have four attributes: a Title, Units, Description, and Definition.
Sysfunction	A Function that is pre-defined in Analytica.

Sysvar	A pre-defined variable that controls the way things are printed, plotted, etc.
Variable	An element of a model that can have a value.
Any	A term used by Analytica to identify a parameter of a Command or a Function that can match any object class. Used internally.

Variable classes

The following classes have a parent class of **Variable**.

Chance *v*	Any variable, usually one defined with an uncertainty distribution, or that evaluates to a distribution of values.
Constant *v*	A variable *v* that has a known, fixed values, e.g., the atomic number of an element.
Decision *v*	A variable *v* represents something that can be chosen or controlled by the decision-maker. A Decision variable cannot be probabilistic.
Determ *v*	Deterministic node type.
Index *v*	A variable *v*, represents something that is a set of scalar values, a dimension along a table.
Objective *v*	The objective, or criterion variable which the model is minimizing or maximizing.

Module classes

The following classes have a parent class of **Module**.

Model *m*	The root model in a model hierarchy, which saves to a file.
Linkmodule *m*	A Module is saved to a file separate from the one used to save its parent model.

Linklibrary *m*	A Library usually contains user-defined functions, and is saved to a file separate from the one used to save its parent model.
Library *m*	A Library usually contains user-defined functions, and is embedded in the model that contains it.
Form *m*	A Form usually contains input nodes and outputnodes, and is embedded in the model that contains it.

Button classes

The following classes have a parent class of **Button**.

Text *f*	A Text node contains arbitrary text.
Picture *p*	A Picture contains a graphic pasted in from another application.

Sysfunctions

Sysfunctions fall under the following categories:

- **Mathematical functions -** functions such as Sine, Cosine, and Tangent.
- **Array functions -** functions for creating and manipulating arrays.
- **Probability Distribution functions -** functions for creating probability distributions.
- **Special functions -** functions used for specialized activities such as sensitivity analysis and dynamic simulation.
- **Statistical functions -** functions for statistical analyses, such as computing the mid value or variance of a sample distribution.

Sysfunctions are summarized in the Analytica Reference.

Sysvars

Sysvars fall under the following categories:

- **Graphing System variables** - Sysvars that control the display of graphs. These are described in the Macintosh Developer's Guide, because they are only available on the Macintosh.
- **Other System variables -** Sysvars for controlling other aspects of the display.

Other System variables

Abbreviation	System variable determining Analytica' ability to understand abbreviations. **Abbr: 1** means abbreviations accepted, **abbr: 0** means abbreviations are not permitted
Prompt	System variable. There are three types of prompts: an Outer-level prompt ("**>**"), a dialog prompt ("**:**"), and a "continue" prompt ("continue:").
Numberwidth	A System variable that controls the width used in printing out numbers. Default is 4. values less than 4 aren't recommended. If NumberWidth is set to 12 or more, numbers will be printed in floating point "E" format (e.g., 1.234456789E+12). The value 0 means "free format", and Analytica will choose what width it likes. *syntax*: **numberw:** *x*
Run	System variable that indexes a sample distribution.
Samplesize	System variable that determines the number of elements in a sample distribution. **samplesize:100** is the default; **samplesize: 32000** the maximum.
Sampletype	System variable specifying the procedure for sampling a distribution, i.e., Median Latin Hypercube

	(Sampletype: 0), Random Latin Hypercube **(Sampletype: 1)**, or Simple Monte Carlo **(Sampletype: 2)**.
Time	System variable, usually used as an index for Dynamic. For dynamic models, this must be assigned a list of the time points (e.g. years) at which the Dynamic variables are to be evaluated.
Checking	System variable telling Analytica to examine all of the Check attributes in a model and to flag the first incidence of a Definition that is in conflict with its Check. **checking:1** switches it on (default); **checking:0** switches it off.
Typechecking	A test that Analytica runs to see if a variable matches its Type assignment. **TypeChecking: 1** switches it on; **TypeChecking: 0** switches it off.
Verbosity	System variable. Controls how "chatty" Analytica is. **verb: 1** verbose; **verb: 2** does not print tables unless explicitly requested; **verb: 4** prints "evaluating <var>" when it is; **verb: 8** is Filetrace; **verb: 16** is debug mode.
Heapsize	Holds the amount of memory needed to evaluate the model, based on the last time it was evaluated. Used to aid Analytica in determining whether enough memory is available to evaluate a model.
It	Used internally by Analytica. "It" refers to the value of the last computed expression.

Syntax

Arithmetic Operators	**+** (plus), **-** (minus), ***** (times), **/** (divide), **^** (to the power of).
Assignment Operator	**:** (Colon).
Comparison Operators	**<** (less than), **<=** (less than or equal to), **=** (equal), **>=** (greater than or equal to), **>** (greater than).
Logical Operators	**And Or Not**
Conditional Operators	**If Then Else**
Comments	Surrounded by "squiggled" brackets.
Arrays	Literal (one-dimensional list) arrays are enclosed in square brackets **[]**. Negative values must be enclosed in parentheses or set off by commas. As the last argument to table, an array is a list enclosed in parentheses.
Command	The name of the command-name is followed by a list of parameters. You do not need parentheses around the parameter list. Parameters may be separated by commas (optional).
Function calls	The name of the function or user-defined function is followed by a list of parameters, enclosed in parens (optionally). Separating commas are optional.

Esoteric/Obsolete/Not Implemented objects

The following is a partial list of esoteric or unimplemented objects. If you are familiar with an object or command that doesn't appear anywhere in this list, or in the above lists of objects, chances are it is obsolete and is no longer implemented.

Bigbrother	Bigbrother controls the file IAC and startup behavior of Analytica. If the value is 0 at startup, Analytica displays an Open Project dialog box, shows a Diagram window when opening the project, and ignores IAC file input (it still recognizes AppleEvent input). If the value is 1 at startup, Analytica does not display an Open Project dialog box, and does not show a Diagram window when opening a project. Analytica responds to IAC file input as well as AppleEvent input.
Dyninputs	A computed attribute for dynamic variables, which contains a list of the variables and functions (at previous Time period(s)) on which the variable depends.
Dynoutputs	A computed attribute for dynamic variables, which contains a list of the variables and functions (at subsequent Time period(s)) which depend on the variable.
Needs	An attribute of a model listing the other models or versions that contain objects that are used by the model. This is used when reading in a model from a file to make sure all other required models are also read in.
Neededby	An attribute of a model listing the other models that need this model.
Fixedcheck	A computed attribute, containing a compiled version of Check.

Fixeddef	A Computed attribute, containing a compiled version of a Definition. Used internally by Analytica.
Paramtypes	The types of the parameters of a function, derived from the Parameters attribute.
Paramnames	The names of the parameters of a function, derived from the Parameters attribute (used internally to re-generate a definition when an identifier changes).
Help	Returns help about a feature in Analytica. Not yet implemented.
Infromrec	Infromrec should never be set by the user. It is a system Function that is used by Analytica to determine whether or not a record file is being read. Analytica prints a call to Infromrec when it reads a record file. The user should never remove any call to Infromrec from a file.
Levels	The number of levels of dependency to display in response to the Tree command.
Makerect *(x)*	To rectangularize a non-rectangular array (not yet implemented).
Normal_fracs	System variable used in calculating normal distributions.
Origcontains	When reading Contains attribute in a model file, a copy is initially stored in this attribute.
Pred	Internal function for “previous time” syntax. *Syntax*: *v*[**time**-1].
Usinglocal	Internal function for “Using … Do” and “For … Do” syntaxes.

Index