Lotus

Approach 96 EDITION

The High-Powered Database
The Whole Team Can Use.

USING LOTUSSCRIPT IN APPROACH

WINDOWS 95

Contents

1 An Introduction to Scripting in Approach 1-1

Scripting vs Other Ways of Automating Tasks 1-1
Approach features that automate tasks 1-1
Macros 1-2
Scripts 1-2
LotusScript 1-2
Getting Help on LotusScript 1-3
Additional LotusScript documentation 1-3

2 Objects, Properties, Methods, and Events 2-1

Objects 2-2
Objects and classes 2-2
Properties 2-3
Methods 2-4
Events 2-5

3 The LotusScript IDE 3-1

IDE Functionality 3-1
Accessing the IDE 3-2
Viewing Classes in the IDE Browser 3-3
Displaying a list of classes, methods, properties, and events 3-3
Pasting a class, method, property, or event name into the current script 3-4
Getting context-sensitive Help in the Browser 3-4
Working in the IDE with Events 3-4
Accessing an object's event 3-6
Sub and End Sub 3-7
Attaching a script to an object event 3-8
Reusing Scripts 3-10
Copying and pasting scripts 3-10
Importing and Exporting a Script 3-11

4 Writing Simple Scripts 4-1

Setting a Property 4-1
Calculating a Value 4-2
Testing Conditions 4-3
Creating a New Object 4-5
Introducing the sub 4-6
Declaring a variable 4-6
Setting the location of the new object 4-7
Setting the properties of the new object 4-7
Creating a Function 4-8
Example 4-9

5 Advanced Scripting Concepts 5-1

Objects as Properties 5-1
Containment 5-2
Classes that Inherit from Other Classes 5-5
Accessing Data 5-8
Accessing Approach data from other Lotus products 5-9
Example 5-9

6 OLE Automation Support 6-1

Using OLE Automation in Lotus Products 6-1
Automation controllers 6-1
Automation servers 6-1
Accessing LotusObjects 6-2
LotusScript applications as controllers 6-3
Object names for applications 6-3

Chapter 1
An Introduction to Scripting in Approach

Lotus® Approach® provides several powerful tools for automating and enhancing Approach tasks. In addition to macros, Approach now has the LotusScript™ language, which lets you develop your own applications as well as automate existing tasks.

Scripting vs Other Ways of Automating Tasks

A script is a sequence of instructions, often a program, that you write to automate a task or customize a product for your specific needs. Any task that you repeat frequently is a candidate for scripting. In fact, the simplest use of scripting in Approach is the automation of a sequence of Approach commands.

Approach features that automate tasks

Not all Approach command sequences require scripting. Approach provides features that enable you to automate certain tasks. For example, the Named Find/Sort feature lets you search for records according to the criteria you define. By saving the search criteria, you can repeat the search any time as needed. You can also easily switch between different found sets of records.

Another example of a feature that automates tasks is the Drill-down to data feature, which allows you to choose data in a crosstab or chart and view the details behind that data. Normally, when you look at a chart or crosstab, you don't actually see individual data; you see groups and calculated summaries, totals, counts, averages, and so on. Drill-down gives you a way to see the data that goes into any of the groups or calculated values, by creating a new worksheet that displays the data. You can add fields to this worksheet and save it with your changes. Then, every time you use the Drill-down to data feature on that crosstab or chart, the worksheet will display as you formatted it.

Macros

Macros are another way of automating frequently performed tasks and are easily built using point and click functionality. Some of the tasks you can automate using macros include the following:

- Switching from one Approach view to another in a file
- Changing or setting data in a field for a small to medium set of records
- Importing or exporting records
- Switching to the next, previous, first, or last record
- Displaying a message box with text and various user choices

Scripts

While macros are very easy to build, some database operations are executed faster with scripts. Scripts are not only fast, they are often more flexible in processing large amounts of data. Use scripting to perform tasks such as the following:

- Changing or setting data in a field for a large set of records
- Changing the attributes (color, size, position, visibility, and so on) of a field or view
- Triggering actions that occur in response to many different events (click, double-click, key press, and so on)
- Displaying or manipulating Approach dialog boxes
- Calling functions

LotusScript

In Approach, you can write scripts and programs using the LotusScript programming language. LotusScript is a standard-BASIC scripting language that is available in most Lotus products and that allows you to write programs across multiple Lotus products. It provides a high degree of flexibility to control the flow of a program. The following are some of the tasks you can perform with LotusScript:

- Write programs that store and manipulate data
- Execute tasks based on the value of a specified condition
- Perform loops (repetitions of processes) if you want to repeat tasks

- Communicate with and pass data to and from other Lotus products that support LotusScript
- Communicate with and pass data to and from any product that uses OLE Automation
- Print text and values

Each Lotus product that uses LotusScript has its own set of LotusObjects™ that can be manipulated with LotusScript. Some objects are similar across products, while some are unique to a product. For more information about objects, see Chapter 2, "Objects, Methods, Properties, and Events."

Getting Help on LotusScript

Help is available for LotusScript. When working in Approach, you can scan Help to find information about the following topics:

- LotusScript functions and statements
- The LotusScript Integrated Development Environment (IDE)
- Approach-specific LotusObjects
- LotusObjects for any other installed Lotus products, besides Approach, that use LotusScript

To access Help about LotusScript, choose Help to open the Help contents and click the book called "LotusScript."

To access Help about the IDE, choose Help to open the Help contents and click the book called "Using the IDE in Approach." For more information about the IDE, see Chapter 3, "The LotusScript IDE."

To access context-sensitive Help in the IDE Browser for any LotusScript function or statement, or any LotusObject, select the name of the language component and press **F1**.

Additional LotusScript documentation

In addition to this manual and online Help, you can refer to the following materials for information on LotusScript:

- The *LotusScript Language Reference*, a comprehensive summary of the LotusScript language, presented in A-Z format. The *LotusScript Language Reference* is available as Help in all Lotus products that support LotusScript and is also available in print. To obtain this manual, use the coupon enclosed in your Approach package.
- The *LotusScript Programmer's Guide*, a general introduction to LotusScript that describes its basic building blocks and how to put them together to create applications. To obtain this manual, use the coupon enclosed in your Approach package.

Chapter 2
Objects, Properties, Methods, and Events

When you write scripts in Approach, as in other Lotus products, you use what are referred to as objects, properties, methods, and events to perform application tasks. This chapter explains the basics of objects and the properties, methods, and events associated with them.

An object represents a part of Approach that you manipulate in a script. For example, the FieldBox object represents a field box. You might want to write a script that checks the data in a field box and performs actions based on the value in the field box. To do this, you use the FieldBox object.

Objects have characteristics, called properties, that allow you to control the way an object looks or acts. For example, some properties of the FieldBox object are Top (defines the top position of a field box), Left (defines the left side position of a field box), Visible (defines whether or not a field box displays on the screen), Font (defines the style of the text inside a field box), and Background (defines the color of a field box).

Objects also have methods, which are actions performed on, or in a few cases by, the objects. The actions change the object or give you control over certain aspects the object's behavior. For example, the FieldBox object has a SetFocus method, which sets the focus from a field box to another object. It also has a Refresh method, which refreshes the value in a field box.

All objects have events. Events are actions performed by a user, an application, or your system. When an event takes place, the script associated with that event is executed. For example, the Click event of the FieldBox object executes whatever script is associated with the event whenever you click the FieldBox object with the mouse.

Objects

The following table introduces some frequently used Approach objects (this is not a complete list):

Object	*Description*
Button	A command button. You can attach a script to a button and run it whenever the user clicks the button.
DropDownBox	A control that displays a list of values in a drop-down box. The user can select only one item in the list.
FieldBox	A control that displays data and allows for data entry. The data may or may not be from a field in a database.
Form	A type of view that displays data one record at a time, like a form at a doctor's office. You can add, edit, and delete data in a form.
ListBox	A control that displays a list of values.
RadioButton	A control that allows selection of only one of a particular set of entries for a field.
Report	A type of view that displays multiple records on one page. You can summarize data in a report.
TextBox	A control that displays text on a form, report, mailing label, form letter, or chart.
Worksheet	A type of view that displays data in a spreadsheet format with each row representing a record, and each column a field. You can add, edit, and delete data in a worksheet.

Objects and classes

An object is technically an instance of a class. A class is a description or a definition of a part of Approach. It has members, which are its properties, methods, and events. For example, Top (a property), Refresh (a method), and Click (an event) are members of the FieldBox class.

A FieldBox object is a specific instance of the FieldBox class. It represents an existing field box in one of your Approach files. You can manipulate it in a script, by referring to its unique object name, for example, Field1. Another instance of the FieldBox class is another FieldBox object with a unique name of its own, for example, Field2.

The easiest way to distinguish between classes and objects is to think of the class as the description of a part of Approach, and the object as one instance of the part described by the class. That instance can be identified with its own unique name, that distinguishes it from all other instances of the class. For example, the word "movie" describes a type of entertainment. "Casablanca" is one instance of the class movie. "Gone with the Wind" is

another instance of that class. “Casablanca” and “Gone with the Wind” are like objects because they are specific, real movies; whereas movie is like a class because it describes the characteristics of a movie without specifically naming any particular movie.

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type this word:

   Classes
3. Select a topic, and then click Display.

Properties

Properties have predefined values that you can change, or “set,” to control the behavior or appearance of an object. For example, in a script you can set the Visible property of the FieldBox object to FALSE, causing the field box to no longer display. Setting the Visible property back to TRUE redisplays the field box.

You can find out what the current value of a property is, by accessing, or “getting,” the property’s value. You set and get property values using scripts.

The following table introduces some frequently used Approach properties (this is not a complete list):

Property name	*Description*
Background	The background color of the object
Color	The color of the object’s background
Left	The location of the leftmost edge of the object
Text	The text value of a selection in an object like DropDownBox, ListBox, or TextBox
Top	The location of the top of the object
Visible	Indicates whether or not the object is visible

Some of these properties apply to a variety of objects. To determine which properties belong to which objects, use the IDE Browser (described in Chapter 3, “The LotusScript IDE”).

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type this word:

   Properties
3. Select a topic, and then click Display.

Methods

The following table introduces some frequently used Approach methods (this is not a complete list):

Method name	*Description*
BringToFront	Makes the object the topmost object.
NextRecord	Goes to the next record.
PrevRecord	Goes to the previous record.
Refresh	Refreshes data.
SendToBack	Makes the object the backmost object.
SetFocus	Sets the focus to the named object.

Some of these methods apply to a variety of objects. To determine which methods belong to which objects, use the IDE Browser (described in Chapter 3, “The LotusScript IDE”).

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type this word:

   Methods
3. Select a topic, and then click Display.

Events

Approach scripts are event-driven This means that in Approach you can initiate script execution by triggering an event. An event is triggered by a user action, an application, or your system.

For example, you can use the Click event of a Button object to initiate execution of a script. When a user clicks the button, the script you associated with the Click event runs. You can control processing and functionality by specifying what operations occur as a result of the event taking place.

You can also initiate execution of a script from a macro or from another script, but the most common way is via an event.

The following table introduces some frequently used Approach events (this is not a complete list).

Event name	*Description*
Click	Occurs when an object is clicked using the mouse. Use when you want the user to initiate an action. For example, you can write a script to change the color of the text in a field box when the user clicks a button.
Change	Occurs when the information in a data entry-type object changes and the object loses focus. For example, if the user selects or deselects a check box, this event occurs.
DoubleClick	Occurs when the object is double-clicked using the mouse.
GotFocus	Occurs when an object gets the focus by being tabbed to, clicked, or selected by the keyboard, a script, or a macro. An object that has the focus may appear bold, highlighted with a dashed border, and so on.
LostFocus	Occurs when an object had focus and then loses it by being tabbed from or by another object being clicked or selected by the keyboard, a script, or a macro.
MouseDown	Occurs when a mouse button is pressed down while the mouse pointer is over an object.
MouseMove	Occurs when the mouse pointer is moved over an object.
MouseUp	Occurs when a mouse button is released while the mouse pointer is over the clicked object.

These events apply to a variety of objects. To determine which events belong to which objects, use the IDE Browser (described in Chapter 3, “The LotusScript IDE”).

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type this word:

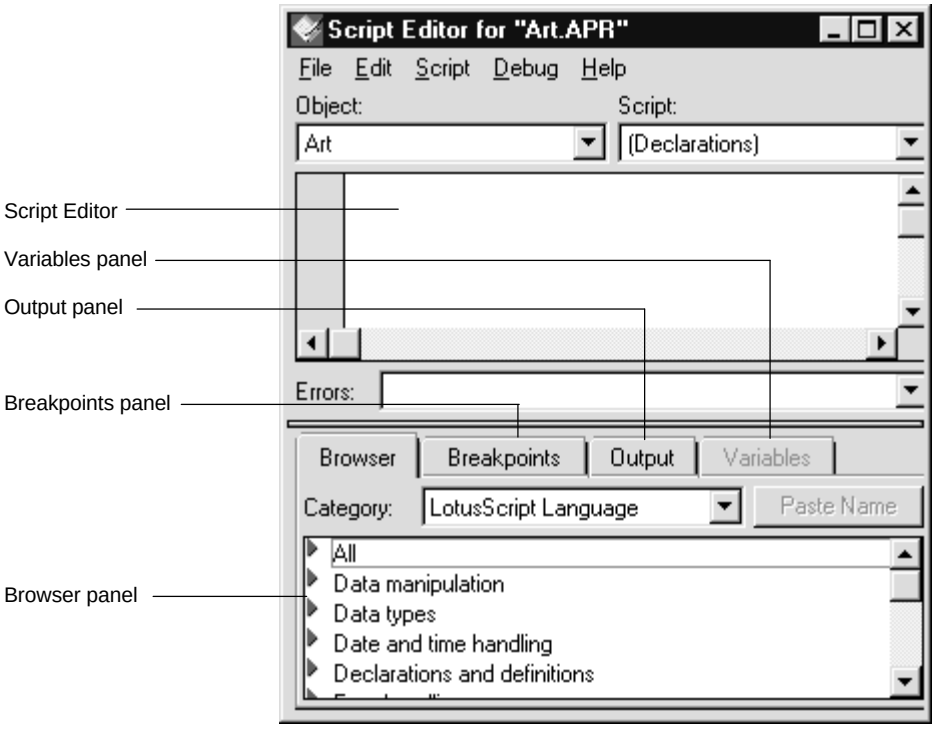
   Events
3. Select a topic, and then click Display.

Chapter 3
The LotusScript IDE

The LotusScript Integrated Development Environment (IDE) is a set of tools you can use in Lotus applications to create and debug scripts.

IDE Functionality

When you open the IDE, a screen like this appears:



The IDE includes the following functionality:

- The Script Editor lets you write scripts and check their syntax. It also lets you set, clear, enable, and disable breakpoints used in debugging scripts.

(A breakpoint is a user-set location in a script where normal execution is interrupted to allow you to check the code. You insert a breakpoint in a place where you think a problem has occurred. The script stops executing at the statement that has the breakpoint, allowing you to check for errors by proceeding through the script one line at a time.)

- The Script Debugger lets you set, clear, enable, and disable breakpoints and step through scripts to locate the source of problems in script execution.
- The Breakpoints, Browser, Output, and Variables panels help you create and debug scripts:
  - The Breakpoints panel lists breakpoints set in scripts, and lets you navigate to breakpoints, as well as clear, enable, and disable breakpoints. (You can also clear, enable, and disable them using the Script Editor.)
  - The Browser panel lists LotusScript keywords, Approach classes and their members (methods, properties, and events), Approach constants, Approach subs and functions, Approach variables, and OLE Automation classes. It also lets you paste keywords, class and member names, and OLE application-class identifiers into a script.
  - The Output panel displays output generated by LotusScript Print statements executed in scripts.
  - The Variables panel displays information during debugging about variables in a script and lets you change the values of variables.

Tip By default, certain language elements are displayed in different colors in the Script Editor and the Script Debugger. For example, keywords are displayed in blue, text containing errors are displayed in red, and user input is displayed in black. To change these default colors, choose File - Script Editor Preferences in the IDE.

Accessing the IDE

1. Choose Edit - Show Script Editor.

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type this word:

   IDE
3. Click the index entry you want, then click Display.

Viewing Classes in the IDE Browser

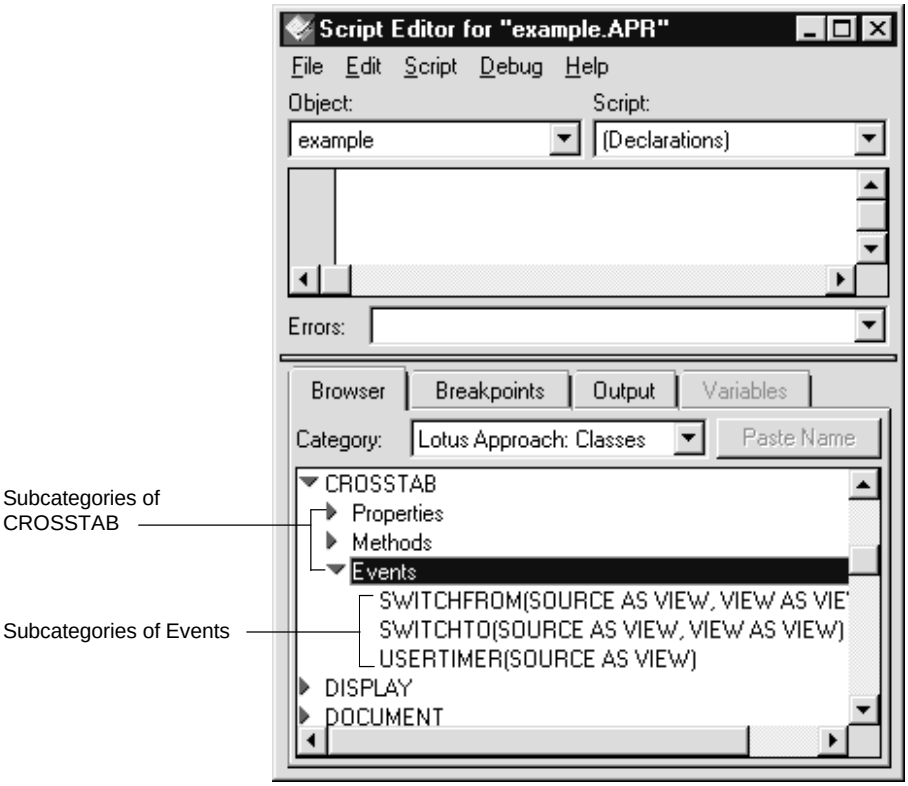
The Browser displays Approach classes and all their properties, methods, and events and lets you paste the name of these language components into the current script.

The Browser panel is available from the Script Editor and the Script Debugger, but you can select items and paste them into the current script only when the Script Editor is active.

Displaying a list of classes, methods, properties, and events

1. In the Category drop-down box, choose Lotus Approach 96: Classes.
2. Click the arrow next to the name of a class to expand it.

   Note A right arrow (▸) means that the item has a list of subcategories that you can expand. A down arrow (▾) means that the list is already expanded.
3. Click the arrow next to Properties, Methods, or Events to expand the subcategory you want to look at.



Pasting a class, method, property, or event name into the current script

1. Select an item from an expanded category.
2. Click Paste Name at the top of the Browser panel.

   The class, method, property, or event name you selected is pasted into the current script at the location of the mouse pointer before you started pasting.

Getting context-sensitive Help in the Browser

1. Select an entry for an Approach class, method, property, event or other LotusScript language component.
2. Press **F1**.

   Help for the component is displayed.

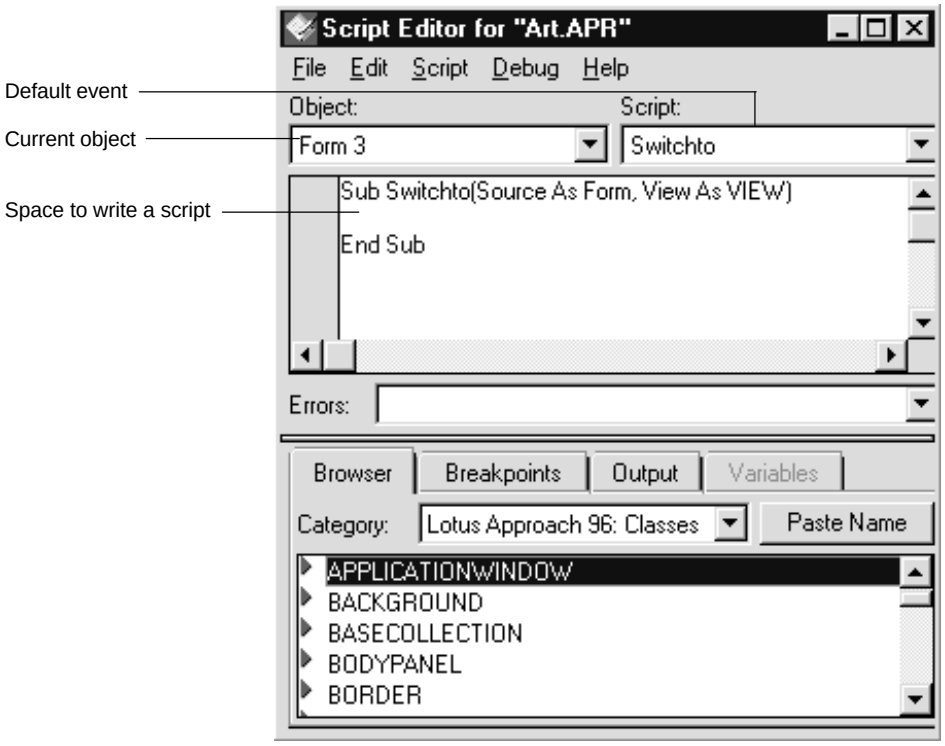
Working in the IDE with Events

All objects have events. Events are the primary way to trigger the execution of scripts. The IDE lets you do the following:

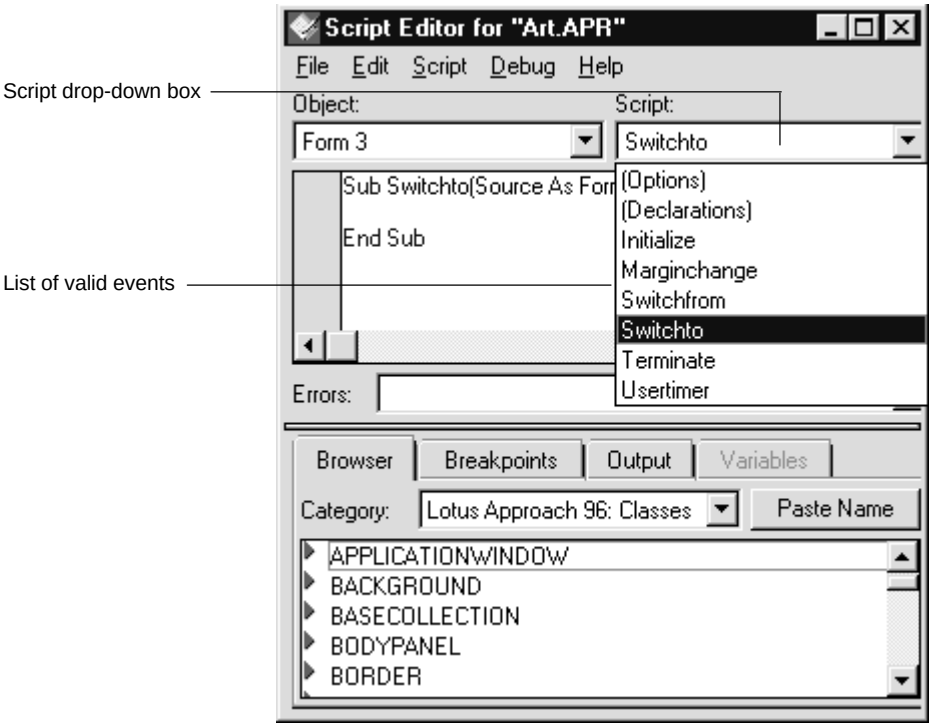
- View a list of all events associated with an object
- View the script associated with an event for the current object
- Write scripts for the current event

When you write a script for an event, you are really writing it for a single object/event pair. The same event name may be associated with another object, but it is still considered a unique event. You can attach a different (or the same) script to each object/event pair. This means that while an event may belong to several different objects, it is the script for a specific object/event pair that is triggered when an event occurs for the object.

When you open the IDE from the main menu, the Script Editor shows the last script edited. If no script was edited, the Script Editor shows the script attached to the default event for the current object. If no script is attached, it shows the space to write a script for the event, such as in the following screen. This screen shows the Switchto event for Form 3, with a space to write a script for the event.



You can access a list of valid events for the current object by clicking the Script drop-down box.



Accessing an object's event

1. Click the Object drop-down box.
2. Select an object from the list.
3. Click the Script drop-down box.

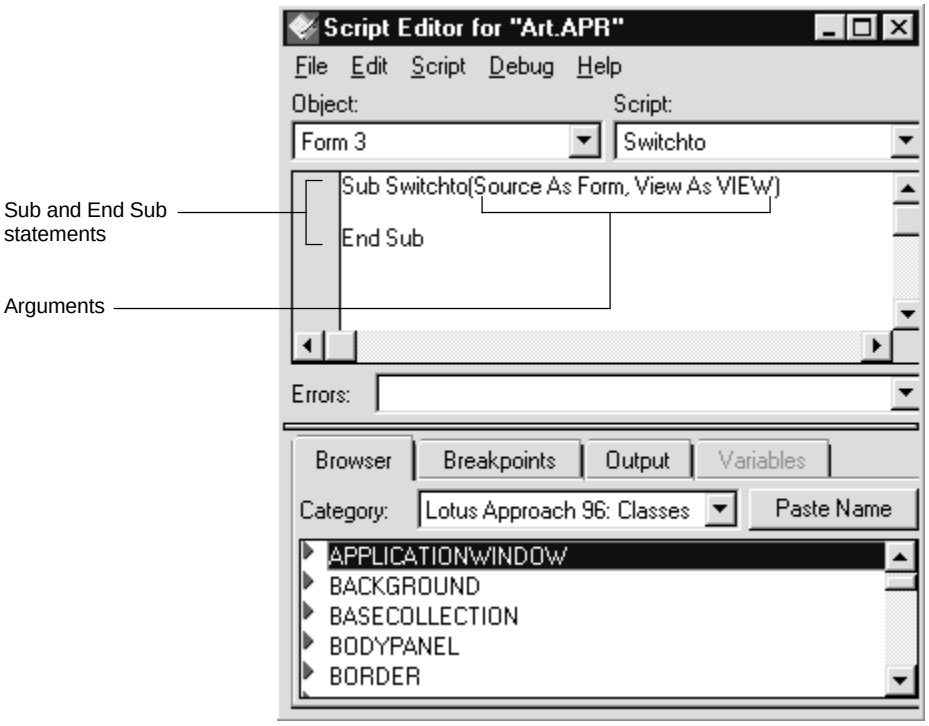
   A list of events for the selected object appears.
4. Select an event.

   The Script Editor displays the script for the object event you selected.

Sub and End Sub

Event scripts are subroutines. Subroutines, or subs, are the parts of a script that perform specific tasks without returning a value (unlike a function, which does return a value). Subs begin with the Sub keyword followed by the name of the sub and the sub's arguments, and end with the line End Sub. The script that you write between the Sub and End Sub keywords determines what the sub does.

Arguments are the information in parentheses next to the name of the sub. An argument is the information that is supplied to the sub so that it can perform its tasks. You specify an argument to provide the sub with a mechanism for accepting data that is coming from elsewhere when the sub is invoked. For example, in the following screen, Source As Form and View As VIEW are arguments that tell the sub which form to switch from and which view to switch to. For more information about arguments, see the *LotusScript Programmer's Guide.*



The purpose of an event is to initiate the execution of the script inside the sub attached to the event. When the event occurs, the sub is executed. By writing a script inside the sub, you are attaching that script to the object event.

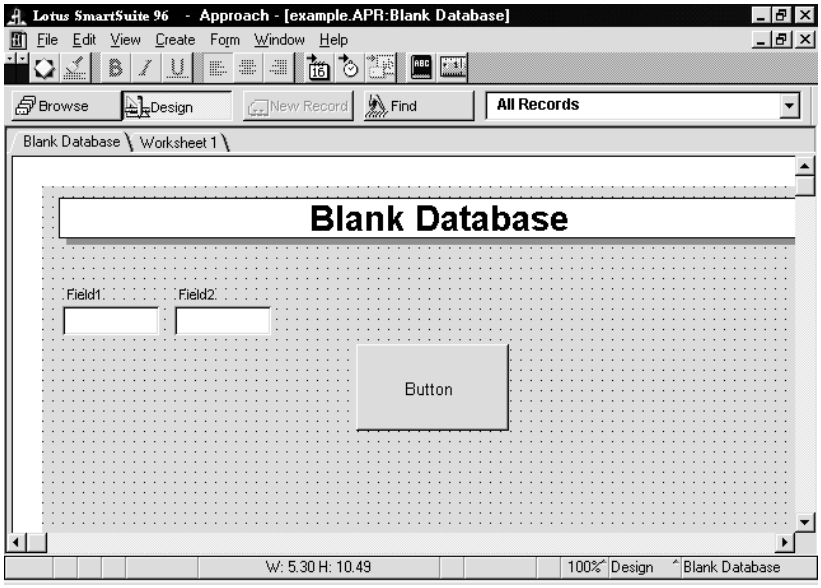
Attaching a script to an object event

The following steps use a simple example to illustrate how to attach a script to an object event.

1. Create a new Approach file (EXAMPLE.APR) with two fields, one called Field1 and the other called Field2, on the body of a form.

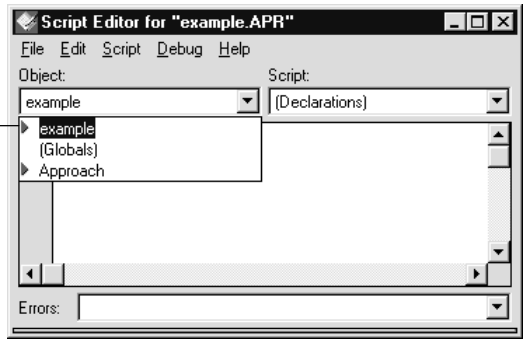
   For information about creating a new Approach file, see Chapter 5, "Creating Databases," in *Exploring Approach.*

2. Choose View - Design to switch to design mode.
3. Choose Create - Control Button and draw a button on the form.



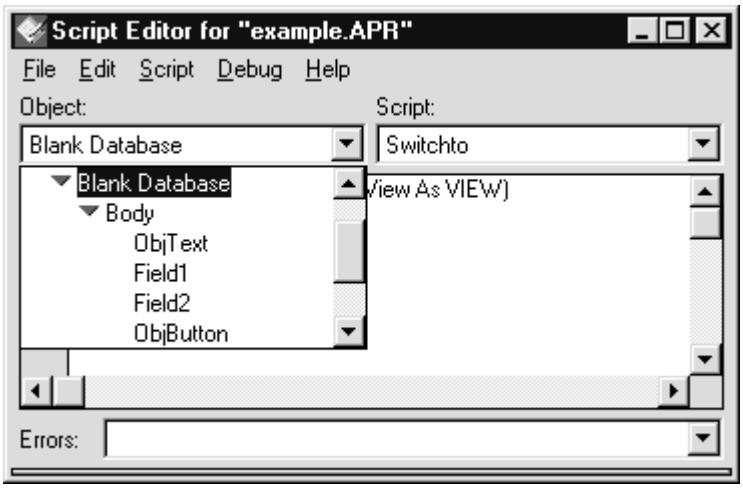
4. Choose Edit - Show Script Editor and click the Object drop-down box.

   The name of the .APR file is the first item on the list.

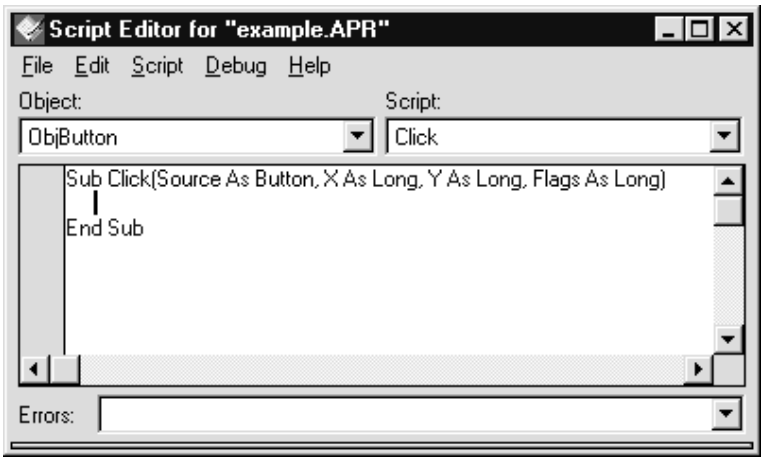


Name of the .APR file

5. To expand the list of objects in the file, click the arrow (▸) next to the name of the file, then click the arrow next to the name of the View object representing a form (called Blank Database, the name assigned by Approach). Then click the arrow next to the name of the Body object (in this case Body).



6. Click ObjButton, which is the name Approach assigned to the Button object you created.



The Click event, the default event for the Button object, appears in the Script drop-down box, while the following code appears in the Script Editor:

```
Sub Click(Source As Button,X As Long,Y As Long,Flags as
Long)


End Sub
```

Click is the name of the sub.

The information in parentheses represents arguments that are passed to the sub. The arguments in this case are Source, X, Y, and Flags.

7. Type the following script between Sub Click and End Sub in the Script Editor:

   ```
   Source.Field1.Text="Hello"
   ```

   This script, entered in the Click event script for ObjButton, changes the text in Field1 to "Hello" when the button is clicked.

   For information about the elements of scripts like this, see Chapter 4, "Writing Simple Scripts."

Reusing Scripts

The IDE saves you time when you want to reuse code. If you have a script attached to an object event and you want to use the same script for another object event, you can copy and paste it.

Copying and pasting scripts

1. Choose Edit - Show Script Editor and click the Object drop-down box.
2. Select the name of the object that has the code you want to copy.
3. Click the Script drop-down box and select an event name.

   The Script Editor displays the script attached to that object event.
4. Highlight the entire script or the portion of the script you want to copy.
5. Choose Edit - Copy in the IDE.
6. Click the Object drop-down box and select the object that has the event you want to copy the script to.
7. Click the Script drop-down box and select the event.

   The Sub...End Sub statement for the event appears.

8. Insert the mouse pointer inside the Sub...End Sub statement and choose Edit - Paste in the IDE.

   The script is pasted in the Sub...End Sub statement.

Importing and Exporting a Script

The IDE lets you import and export scripts into other file formats so that you can open and edit your scripts in other products, and then bring them back into Approach easily. This feature is particularly useful if you prefer to write and edit your scripts in an editor other than the one provided by the IDE.

The File - Import Script and File - Export Script commands allow you to import and export scripts to and from text files, and to and from other Lotus products besides Approach.

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type one of these words:

   Importing

   Exporting
3. Click the index entry you want, then click Display.

Chapter 4
Writing Simple Scripts

This chapter uses examples to introduce some basic concepts that will help you to write scripts in Approach.

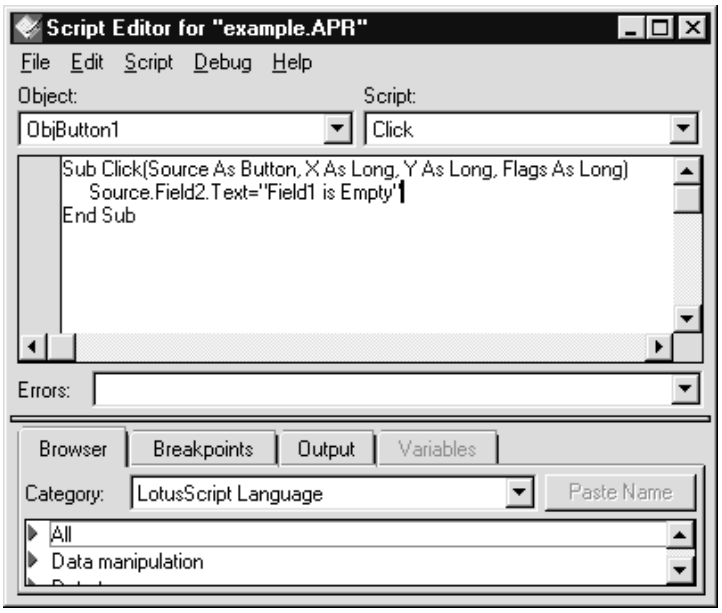
Setting a Property

The following script is associated with the Click event of a Button object, called ObjButton1, which is on a view that has two fields: Field1 and Field2. The script runs every time the button is clicked.

Note An example in Chapter 3 uses a file called EXAMPLE.APR to explain how to attach a script to an object event. The following script builds on that example, using the same file and another button object (Approach assigns it the name ObjButton1) on the form called Blank Database.

To place the words “Field1 is Empty” in Field2, use the following statement:

```
Source.Field2.Text = "Field1 is Empty"
```



Source refers to the current object, in this case ObjButton1.

Field2 is the object whose property is being changed.

Text is the property being changed. It has a data type of String, which means that it only accepts values that are text.

"Field1 is Empty" is the value that the property is being set to. The value is a text string.

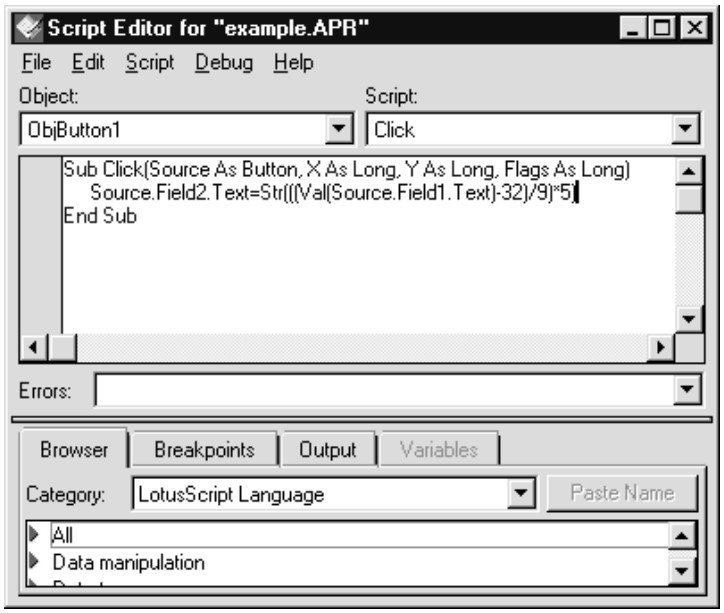
Note Using dots between identifiers in these code examples is referred to as dot notation. You use the dots to connect objects with their properties and methods, and to connect objects with objects they are related to. For information about containment and inheritance relationships between objects, see Chapter 5, "Advanced Scripting Concepts."

Calculating a Value

The following script is associated with the Click event of a Button object, called ObjButton1, which is on a view that has two fields: Field1 and Field2. The script runs every time the button is clicked.

This example makes a calculation based on a formula and places the result in Field2. The formula converts the value entered in Field1 to centigrade.

```
Source.Field2.Text=Str(((Val(Source.Field1.Text)-32)/9)*5)
```



Source refers to the current object, in this case ObjButton1.

Field2 is the object whose property is being changed.

Text is the property being changed. It has a data type of String, which means that it only accepts values that are text.

Val is a LotusScript function that converts text to numbers. In this example, the function converts the value of the Text property of Field1 so that the value can be used in a mathematical formula.

Str is a LotusScript function that converts numbers to text. In this example, the function converts the numeric value, after it has been calculated in the centigrade formula, back to text so that it can be assigned to the Text property of Field2.

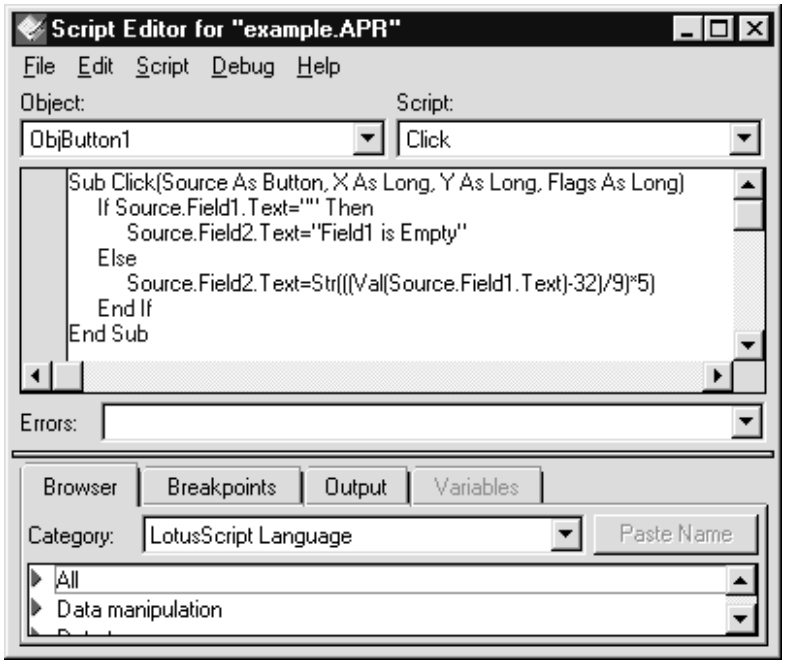
Testing Conditions

This example uses a basic If...Then...Else statement to determine if the centigrade formula should be calculated and its result placed in Field2.

Assume that the script is associated with the Click event of a Button object, called ObjButton1, which is on a view that has two fields: Field1 and Field2. The script runs every time the button is clicked.

```
If Source.Field1.Text="" Then
   Source.Field2.Text="Field1 is Empty"
Else
   Source.Field2.Text=Str(((Val(Source.Field1.Text)-32)/9)*5)
End If
```

Note Do not type a space between the quotation marks ("").



If the text property of Field1 does not contain text, in other words if it contains an empty string represented by the empty quotation marks (""), then the script places the phrase “Field1 is Empty” in Field2. If the text property of Field1 has a value other than the empty string, then the script calculates the formula using that value, and places the result in the Text property of Field2.

The If...Then... Else statement conditionally executes one or more statements. If the condition of the If statement is true, then any statements following Then are executed. If the condition is false, any statements following Else are executed. This statement is very useful for building flexibility into a script. Make sure that you use the same structure as in the example above: the keyword Then must be on the same line as the keyword If in order for the statement to work. The If...Then... Else statement must end with the keywords End If to indicate the end of the condition being evaluated.

Creating a New Object

The example in this section illustrates how a script creates a new object using the LotusScript New keyword in a Set statement. In addition to how it is used in the following example, New can create new objects in the following ways:

- You can use the New keyword in a Dim statement.
- Most Approach classes use New as a means to construct a new object of that class, and treat New as a method. The arguments vary from class to class.

For more information on using the New keyword in the Set or Dim statement, see the *LotusScript Language Reference*.

The following script uses a Approach file, EXAMPLE.APR, created in an example in Chapter 3. The script is attached to the Click event of ObjText, a TextBox object.

```
Sub Click(Source As TextBox, X As Long, Y As Long, Flags as
Long)
   Dim fb As FieldBox
   Set fb=New FieldBox(Source.Body)
   fb.Left=1440
   fb.Top=1440
   fb.Width=2880
   fb.Height=720
   fb.NamedStyle="Default"
   fb.Background.Color.SetRGB(COLOR_GREEN)
End Sub
```

This script does the following when the TextBox object it is attached to is clicked:

- Creates a new field box.
- Sets its left and top sides to 1 inch (1440 twips) from the left and top of the form. A twip (TWentIeth of a Point) is a measurement equal to 1/1440th of an inch.
- Sets its width to 2 inches (2880 twips).
- Sets its height to a 1/2 inch (720 twips).
- Provides borders.
- Sets the background color of the field box to green.

Introducing the sub

The first line of the example identifies the event that causes the script to execute. When the user clicks the object that the event belongs to (a text box), the script executes, creating the FieldBox object.

```
Sub Click(Source As TextBox,X As Long,Y As Long, Flags as Long)
```

Click is the name of the event.

Source refers to the current object, in this case a TextBox object.

As TextBox specifies the data type of Source. A data type is the classification, or category, of a piece of data. It determines whether the data is composed of numbers, text, or an object. The data type of an object is always its class. In this case, As TextBox means that the object has a data type of TextBox.

X and Y refer to the x-coordinate and the y-coordinate of the position of the mouse pointer when you click the current object.

As Long refers to the data type of X and Y.

Flags refers to which mouse button was pressed, the left or the right.

As Long refers to the data type of Flags.

Declaring a variable

The next line of the example declares a variable. A variable is a uniquely named container for storing a piece of data. Every variable has a data type, which determines what kind of value (text, numbers, or an object) the variable is allowed to hold.

When you declare a variable, you specify its data type and in some instances, its value. It's good practice to put all declarations at the beginning of a script. That way you always know where to look for information about variables when reading the code. Use the Dim statement, or one of its variations, to explicitly define a variable's type. For more information about variables, see the *LotusScript Programmer's Guide.*

In the example, fb is a variable with a data type of FieldBox. This means that the only kind of data that can be stored in fb is a FieldBox object. The variable is declared here so that it can be initialized to the FieldBox object created in the next line of the example. Initializing a variable means that you store a value in it.

Setting the location of the new object

The next line of the example does three things simultaneously:

- Creates a new FieldBox object using the LotusScript New keyword.
- Initializes the variable fb to the new FieldBox object. Essentially, the new FieldBox object is stored in fb using the Set statement.
- Places the new FieldBox object in the current Body object (source.body).

```
Set fb=New fieldbox(source.body)
```

Note By placing the FieldBox object in the current Body object, the Body object becomes the "container" object. For information about containment, see Chapter 5, "Advanced Scripting Concepts."

Setting the properties of the new object

The remainder of the script sets the properties of the new FieldBox object stored in fb.

The following two lines of code set the Left and Top properties of the FieldBox object to 1 inch (1440 twips), which positions the FieldBox object one inch from the left and top of the form.

```
fb.Left = 1440
```

```
fb.Top = 1440
```

The next two lines set the width of the FieldBox object to 2 inches (2880 twips), and the height to a 1/2 inch (720 twips).

```
fb.Width = 2880
```

```
fb.Height= 720
```

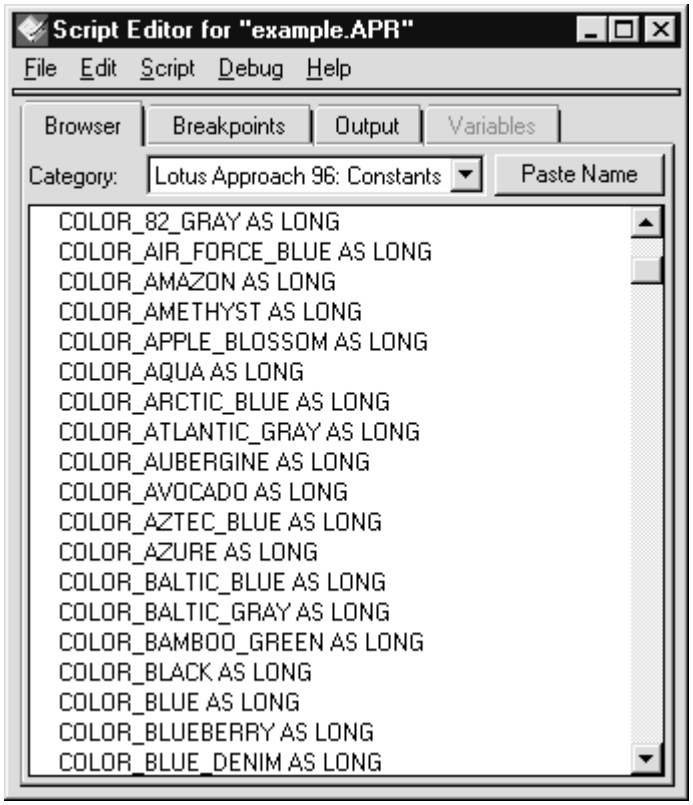
The next line sets the NamedStyle property, which controls the appearance of the FieldBox object, to the Default style. The Default style turns on the borders of the field box (otherwise they'd be invisible). For a list of the acceptable values of the NamedStyle property, see Help.

```
fb.NamedStyle="Default"
```

The last line before the end of the example sets the background color of the field box to green. COLOR_GREEN is the RGB (Red Green Blue) value for green. RGB values determine the colors of objects. Approach has defined a set of RGB values as constants (you can define your own colors as well). A constant is a value that remains unchanged throughout a script, as opposed to a variable, which can change.

```
fb.Background.Color.SetRGB(COLOR_GREEN)
```

A list of the Approach color constants is available in the “Lotus Approach96: Constants” category on the Browser Panel of the IDE. The following screen shows a partial list:



Creating a Function

A function is a procedure in a script with a name assigned to it. It differs from a sub in that it returns a value. You provide values to the function, it processes them, and then sends back a single value result to the procedure that called it.

LotusScript provides a set of built-in functions that you can use to perform many common numeric, date/time, string, data-conversion, and value-testing operations.

You can also create your own functions in Approach. You define a function by specifying a series of one or more statements that are to be executed as a block when you call the function in a script. You enclose these statements between the Function keyword (followed by the function name) and the End Function keyword.

To use a function in a script, simply provide the function name in a line of code in the script, at the point where you want the function to execute, and pass values (called arguments) to it. The function receives these arguments, processes them, and returns a value.

For more information about functions, see Chapter 4, "Procedures, Functions, Subs, and Properties," in the *LotusScript Programmer's Guide.*

Example

The following example illustrates a user-defined function. The function retrieves a value, representing temperature, from a field box and runs the value through one, two, or three tests, depending on what the results of first and second test are:

- If the temperature is less than or equal to 40 degrees, it returns a value that represents the color pale blue.
- If after the first test, the function determines that the temperature is greater than 40 degrees, it checks to see if it is less than or equal to 80 degrees; if so, it returns a value that represents the color green.
- If after the second test, the function determines that the temperature is greater than 80 degrees, it returns a value that represents the color red.

Later, the function is called in the event script for the Change event of a FieldBox object. When the function is called, its actions are performed.

Entering the function in the IDE

To enter the function in the IDE, do the following:

1. Click the Object drop-down box in the IDE.
2. Select (Globals) from the list.

   (Globals) is listed immediately before the list of Approach objects.

3. Type the following code in the Script Editor:

```
Function TempColor (Temperature As Integer) As Long
   If Temperature <=40 Then
     TempColor=COLOR_PALE_BLUE
   Else
     If Temperature <=80 Then
        TempColor=COLOR_GREEN
     Else
        TempColor=COLOR_RED
     End If
   End If
End Function
```

What the script does

The first line of the example introduces the function definition.

```
Function TempColor (Temperature As Integer) As Long
```

TempColor is the function name.

Temperature As Integer is the function argument. It specifies the variable Temperature, to be used by the function for receiving the value that the function is expecting in order to perform its calculation. This statement also specifies the data type (Integer) of the variable. The value of this variable is given to the function by the event script that calls the function.

As Long specifies the data type of the function's return value.

Note Everything in the example between this first line and the End Function line is the heart of the function, as it specifies what the function does.

The next several lines of the example check the value in Temperature and, depending on what the value is, the function returns an RGB color value.

```
If Temperature <=40 Then
   TempColor=COLOR_PALE_BLUE
Else
```

If the value in Temperature is less than or equal to 40, the function returns the color pale blue. Otherwise, if the value in Temperature is greater than 40, proceed to the next line.

```
If Temperature <=80 Then
   TempColor=COLOR_GREEN
Else
```

If the value in Temperature is greater than 40 (already determined by the previous If...Then...Else test) but less than or equal to 80, then the function returns the color green. Otherwise, if the value in Temperature is greater than 80, proceed to the next line.

```
         TempColor=COLOR_RED
      End If
   End If
End Function
```

Notice that you can put an If...Then...Else statement inside another If...Then...Else statement. This is referred to as nesting If...Then...Else statements.

If none of the previous conditions are met, in other words if Temperature is greater than 80, the function returns the color red and then ends.

All If...Then...Else clauses require corresponding End If clauses. All function definitions end with an End Function clause.

In this example, the function is called from inside the Change event script for the field box that contains the temperature value. This means that when the value inside the field box changes and the field box loses focus, for example if a user enters a new value in the field box and then tabs out of the field box, the Change event sub executes causing the TempColor function to be called.

The following script illustrates how the function is called from inside the Change event script for the FieldBox object.

```
Sub Change(Source As FieldBox)
   Dim x As Long
   x=TempColor(Val(Source.Field1.Text))
   Source.Field1.Background.Color.SetRGB(x)
End Sub
```

This script does the following:

- Converts the value of the Text property for Field1 from a string to a number, using the Val function
- Calls the TempColor function and sends it the converted value from Field1 for processing
- Provides the variable x to store the value returned from TempColor
- Sets the background color of Field1 to the value in x, using the SetRGB method

Chapter 5
Advanced Scripting Concepts

This chapter introduces some advanced scripting concepts, which provide you with more powerful capabilities for working in Approach.

Objects as Properties

As explained in Chapter 2, objects have properties, which represent characteristics or attributes of the objects. Some objects also act as properties of other objects. For example, Background, Font, Color, Border, LineStyle, Table, Query, and Connection are objects that are sometimes treated as properties of other objects. The following example illustrates how this works.

Suppose that the current object in a form is FirstName, a FieldBox object. If you want to change its background color to red, use the following statements. (For this example, you must write these statements in an event script, such as the Click event script, of the current object. For information about writing event scripts, see "Working in the IDE with Events," in Chapter 3.)

```
Dim red As New Color(255,0,0)

Set Source.Background.Color = red
```

The first line of the script creates a new Color object called Red and initializes it to red, as indicated by the RGB values 255 (red), 0 (Green), 0 (Blue).

The second line changes the Color property of the Background object to red, using the new object that you created and stored in the variable red, in the first line of the script. The Background object acts as a property of FirstName (the FieldBox object represented by Source), and the Color object acts as a property of the Background object.

Note Whenever you set an object to another object, as in this case where you are setting Background.Color to the Red object, you must use the Set statement. For more information on using the Set statement, see the *LotusScript Language Reference*.

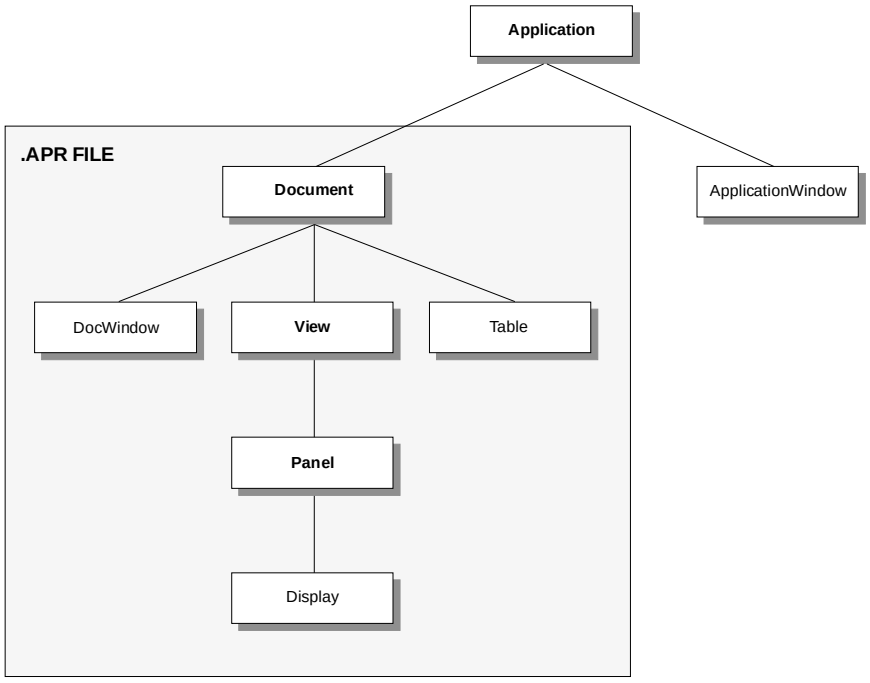
Containment

Some classes contain other classes. For example, the FieldBox class contains the Background class. This relationship is often referred to as a parent-child relationship. Background is a child of FieldBox, which means that FieldBox is its parent. Likewise, the FieldBox class has several different parents including the BodyPanel class, the HeaderFooterPanel class, and the SummaryPanel class. The BodyPanel class has several parents too, including the Form class, the FormLetter class, and the Report class. The Form class has a parent called the Document class.

Note A class can be contained by multiple classes. For example, several different classes, including BodyPanel, SummaryPanel, and HeaderFooterPanel, contain the FieldBox class. This means that the FieldBox class has multiple parents, which are all classes directly above it in the containment hierarchy (see the diagram that follows). However, an object, a single instance of a particular class, can only have a single parent, and that parent must be an object of a class that contains the FieldBox class.

Understanding which classes are contained by other classes, and therefore which objects are contained by other objects, helps you understand the syntax to use in a script when you try to access an object or change one of its properties. One of the advantages of containment is that it lets you access any object by traversing the containment hierarchy that connects objects with other objects.

The following diagram illustrates the containment relationships of classes and objects.



Classes in bold contain other classes. Classes that are not bold do not contain other classes.

Display, Panel, and View are classes that act as categories of other classes. You cannot create an instance (object) of the Display, Panel, or View class. You can, however, create an instance of one of the classes in each of these categories. For example, you can create a TextBox object using the TextBox class, which is a kind of Display class. The following table lists the kinds of Display, Panel, and View classes.

Class	*Kinds of classes*
Display	Button, CheckBox, DropDownBox, Ellipse, FieldBox, LineObject, ListBox, OLEObject, Picture, PicturePlus, RadioButton, Rectangle, RoundRect, TextBox
Panel	BodyPanel, HeaderFooterPanel, RepeatingPanel, SummaryPanel
View	ChartView, CrossTab, Envelope, Form, FormLetter, MailingLabels, Report, Worksheet

Classes (and, therefore, objects) of the Display class category are contained by classes of the Panel class category, which in turn are contained by classes in the View class category, which in turn are contained by the Document class, which in turn is contained by the Application class. While Application is the immediate parent of Document, it indirectly contains the View, Panel, and Display classes. The same is true of Document: it is the immediate parent of View, but it indirectly contains Panel and Display.

Note The ApplicationWindow class is contained by the Application class, and the DocWindow and Table classes are contained by the Document class. However ApplicationWindow, DocWindow, and Table do not contain any classes.

Building on the example in the previous section "Objects as Properties," suppose you want to change the background color of LastName, the name of a FieldBox object. Assume that FirstName, not LastName, is the current object. Since you want to change LastName, you have to access it. You can do this several different ways, using the Approach containment hierarchy.

- If FirstName and LastName have the same immediate parent, a BodyPanel object on a Form object, use the following code to change the background color:

  ```
  Set Source.LastName.Background.Color = red
  ```

  Since FirstName and LastName are siblings with the same immediate parent, a reference to the parent is not necessary. Source, which represents FirstName, is sufficient. This is true for all objects that share the same immediate parent.

- If FirstName and LastName do not share the same immediate parent, for example, if FirstName is contained by Form1, and LastName is contained by Form2, you can traverse up the hierarchy tree starting from the current object (FirstName) and then back down to LastName:

  ```
  Set Source.Body.Form1.Customer.Form2.Body.LastName.
  Background.Color = red
  ```

  Source represents the current object, FirstName.

  Body is the BodyPanel object for Form1, and is contained by Form1.

  Form1 is the Form object that represents the view containing FirstName, and is contained by Customer.

  Customer is a Document object. It is an Approach .APR file, and is the object that FirstName and LastName have in common as a parent (Customer contains both of these objects). At this point you need to refer back down the hierarchy to LastName; you always start back down the hierarchy at the common parent.

Form2 is the Form object that represents the view containing LastName. Form2 is contained by Customer.

Body is the BodyPanel object for Form2, and is contained by Form2.

LastName is the object you are trying to change to red. It is contained by the Body object of Form2, and has the property Background, which is an object that has a property called Color, which is also an object.

- Another option for accessing LastName, if it is not the current object and does not share the same parent with the current object, is to refer to it by starting at the top of the object hierarchy. For example:

  ```
  Set CurrentApplication.Customer.Form2.Body.LastName.
  Background.Color = red
  ```

  CurrentApplication represents Approach, which is the object at the top of the entire hierarchy.

  Customer is the name of the Document object (an .APR file) that contains the form.

  Form2 is the Form object.

  Body is the BodyPanel object for Form2 and is contained by Form2.

  LastName is the object itself and is contained by Body. It has a property called Background which is an object that has a property called Color.

- Instead of the preceding option, you may use the following script to access LastName:

  ```
  Set CurrentDocument.Form2.Body.LastName.Background.Color =
  red
  ```

  In this case, CurrentDocument is equivalent to CurrentApplication.Customer in the preceding example (**`Set CurrentApplication.Customer.Form2.Body.LastName. Background.Color = red`**). CurrentDocument refers to the current Document object in the current session of Approach.

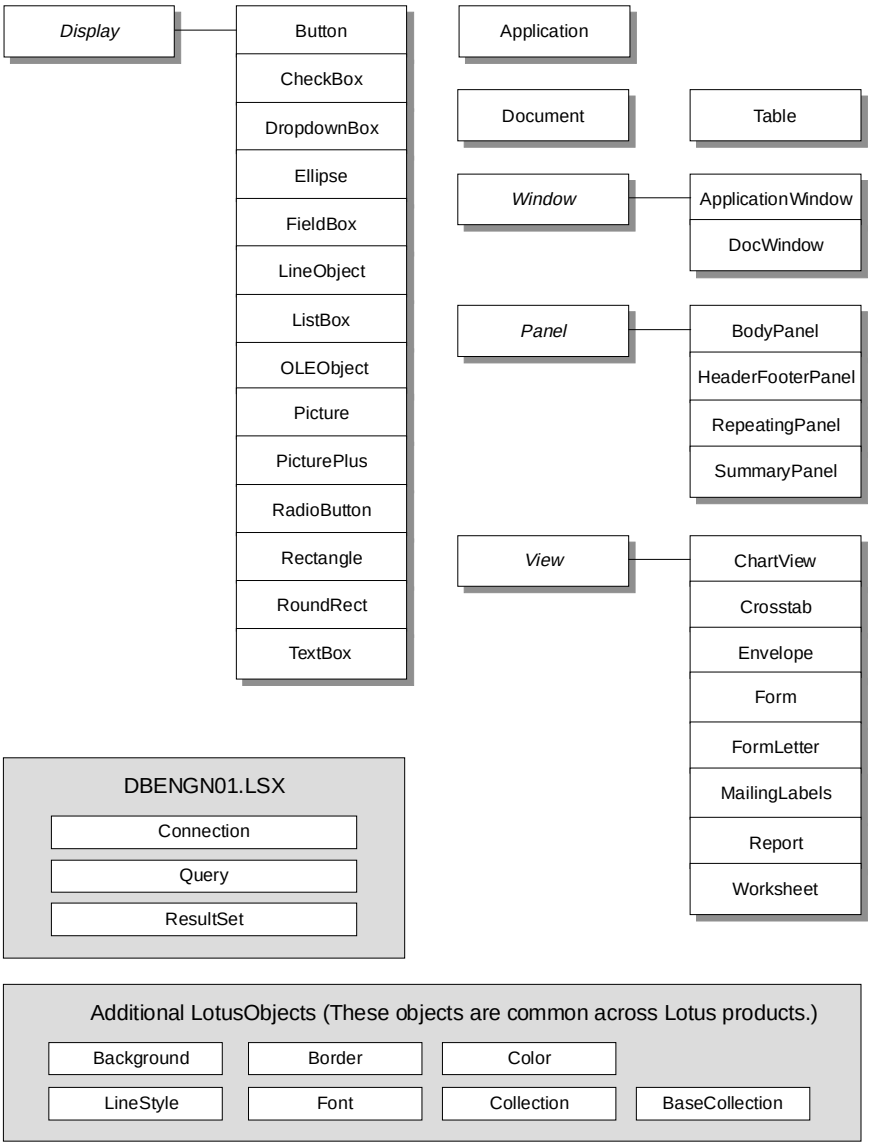
Classes that Inherit from Other Classes

Approach provides some classes, called abstract classes, that exist only to create other classes, called derived classes. You cannot create an instance (object) of an abstract class.

A derived class, also called a subclass, inherits the members (methods and properties) of the class it derives from, called its base class.

The following diagram shows the Approach class hierarchy, including the abstract classes (in italics) and the derived classes that inherit from them.

The table that follows lists the abstract classes and their derived classes.



Abstract class	*Derived classes*
Display	Button, CheckBox, DropDownBox, Ellipse, FieldBox, LineObject, ListBox, OLEObject, Picture, PicturePlus, RadioButton, Rectangle, RoundRect, TextBox
Panel	BodyPanel, HeaderFooterPanel, RepeatingPanel, SummaryPanel
View	ChartView, CrossTab, Envelope, Form, FormLetter, MailingLabels, Report, Worksheet
Window	ApplicationWindow, DocWindow

The advantage of abstract classes is that when the base class of a derived class is an abstract class, you can refer to that abstract class in a script, and then use any of its derived classes in its place, since those derived classes inherit all the methods and properties of the abstract class. This makes the script flexible and reusable.

The following script demonstrates how you can create a sub that accepts an argument with the data type of any class based on a certain abstract class. This example uses the MakeInvisible sub to hide from view an instance of any class derived from the Display class. Display class is an abstract class.

```
Sub MakeInvisible (dis As Display)

   dis.Visible = FALSE

End Sub
```

This sub takes an argument, dis, which has a data type of the Display class. This means that the sub accepts an argument with the data type of any classes derived from the Display class. Button, CheckBox, DropDownBox, Ellipse, FieldBox, LineObject, ListBox, OLEObject, Picture, PicturePlus, RadioButton, Rectangle, RoundRect, and TextBox are all the classes derived from the Display class. You can pass objects of any of these class types to the sub, by assigning the object to the variable dis, and make them invisible. The advantage of this functionality is that it saves you from having to create a different sub for each of the derived classes. Instead, you create one sub that works for all of them.

Keep in mind, however, that when you pass an argument with the data type of a derived class to a function or sub that manipulates properties and uses methods, those properties and methods must be valid for the derived class. For example, the FieldBox class has an Alignment property, but the ListBox class doesn't. The following sub, which left-justifies an object using the Alignment property, would fail if you passed to it a ListBox object as an argument. It, therefore, checks to see if the Type property of the object stored in the variable dis contains the value $aprFieldBox, meaning that it is an Approach FieldBox. If it is, the sub left-aligns the object by setting its Alignment property to $LTSAlignmentLeft.

```
Sub ChangeAlignment (dis As display)
If dis.Type = $aprFieldBox Then
   dis.Alignment = $LTSAlignmentLeft
End Sub
```

For more information

1. Choose Help - Help Topics and click the Index tab.
2. Type one of these words:

   Classes

   Display

   View

   Panel

   Window
3. Click the index entry you want, then click Display.

Accessing Data

Approach provides three objects that let you access data: Connection, Query, and ResultSet. Using these three objects provides you with the power to directly access information in databases and data files (stored in various formats including dBASE®, Paradox®, FoxPro®, Lotus Notes®, and SQL) without going through the Approach user interface (the user interface (UI) refers to the windows, menus, dialog boxes, and various other controls that make up Approach).

The Connection, Query, and ResultSet objects let you access or even delete data via scripting. Using them also allows you to avoid the restrictions of the Approach UI. For example, if you use these objects in a script to manipulate a field defined in the UI to accept values within a certain range, the script doesn't recognize this UI restriction. This means you have a fast and efficient way to manage data, since you aren't restricted by the normal operations defined in the UI.

Caution When you use the Connection, Query, and ResultSet objects, be extremely careful that you don't unintentionally delete data, since it won't be obvious from the user interface what you have done until later, when you try to open an updated file.

Accessing Approach data from other Lotus products

The Connection, Query, and ResultSet objects can be made available to other Lotus products that use LotusScript, so that programs written in those products can access Approach data. These objects are exposed to other Lotus products via a Lotus .LSX file (DBENGN01.LSX) that combines the Connection, Query, and ResultSet objects. An .LSX file is like a dynamically linked library (.DLL) that gives you access to data.

For example, if Lotus Word Pro™ users want to use the Connection, Query, and ResultSet objects to access Approach data, they must do the following:

1. Click the Object drop-down box in the IDE.
2. Select !Globals from the list.
3. Type the following line of code in the Script Editor:

   ```
   Uselsx "[drive name]\[path name]\Dbengn01.lsx"
   ```

   Drive name refers to the drive in which the Approach DBENGN01.LSX file is stored.

   Path name refers to the directory path in which the Approach DBENGN01.LSX file is stored. The default is LOTUS\APPROACH.

Example

This example illustrates the use of the Connection, Query, and ResultSet objects in Approach. The script runs a query against a dBASE IV® table file named EXAMPLE.DBF and prints to the IDE Output panel a list of the table column names. You can type this example in an event script for any object.

```
Dim fName As String
Dim con As New Connection
Dim qry As New Query
Dim rs As New ResultSet
If (con.ConnectTo ("dbase IV")) Then
  Set qry.Connection = con
  qry.Tablename = "C:\Lotus\Approach\Example.dbf"
  Set rs.Query = qry
  If ((rs.Execute)) Then
    N = rs.NumColumns
    Print "Number of columns = ", N
    For i = 1 To N
      fName = rs.FieldName (i)
      Print fName
    Next
  Else
    MessageBox "The query did not run successfully.",0,
"Query execution"
  End If
  con.Disconnect
End If
```

Declaring variables

The first four lines of the example declare and initialize four variables.

```
Dim fName As String
```

This Dim statement declares a variable that will be used later in the script to print the names of the columns in a result set table.

```
Dim con As New Connection
Dim qry As New Query
Dim rs As New ResultSet
```

These Dim statements declare three variables that are essentially initialized to new objects using the LotusScript New keyword. For more information on using the New keyword in the Dim statement, see the *LotusScript Language Reference*.

con is a Connection object variable, which represents a link to a database type.

qry is a Query object variable, which represents the definition of the query (the criteria) for selecting the data you want to retrieve.

rs is a ResultSet object variable, which represents the data returned by the query in column and row format.

The data types of these three variables are their classes.

Testing the connection to the database

The next line of the example introduces an If...Then...Else statement that tests whether the dBASE IV connection is active. To check the condition, the ConnectTo method returns a value that indicates whether or not the database was connected.

```
If (con.ConnectTo ("dBASE IV")) Then
```

If the condition is true, then the script performs several operations.

Setting the query connection

If the dBASE IV connection is active, in other words if the Connection object is a dBASE IV connection, the script sets the Connection property of the Query object to the Connection object (stored in the variable con). This means that the Query object's Connection property is now a dBASE IV connection.

```
Set qry.Connection = con
```

Note Whenever you set an object to another object, as in this case where you are setting qry.Connection to the con object, you must use the Set statement. For more information on using the Set statement, see the *LotusScript Language Reference.*

The script now sets the TableName property of the Query object to the name of the dBASE IV.DBF data file. Approach already knows that the connection is a dBASE IV connection but doesn't know what dBASE IV table to run the script against, so this line of code provides that information.

```
qry.TableName = "C:\Lotus\Approach\EXAMPLE.DBF"
```

Note You must provide an explicit path to the .DBF file. This example assumes that the file is stored in the C:\LOTUS\APPROACH directory.

The following statement sets the Query property of the ResultSet object to qry. A Query object has two properties, a TableName and a Connection, that were just set. These properties designate what data is retrieved and from where.

```
Set rs.Query = qry
```

Running the query

The next line of the example executes the query and verifies whether or not it executed successfully. The Execute method runs the query and creates a result set.

If rs.Execute is successfully executed, then several actions, described next, take place following the If...Then portion of the If...Then...Else statement, which is nested within another If...Then...Else statement.

```
If (rs.Execute) Then
```

If rs.Execute is not successfully executed, then a message is displayed to the user and the script terminates (see below). The Execute method will not be invoked if the file does not exist or the file's path name is not typed correctly.

Iterating through the table

The next six lines of the example determine how many columns are in the table, step through the table using that information, and print the name of each column in the IDE Output panel (for information about the Output Panel, see Chapter 3, "The LotusScript IDE"). The table is represented by rs, the ResultSet object.

This statement captures the value of the NumColumns property of the ResultSet object (rs), and places it in the variable N. NumColumns contains the number of columns in the result set.

```
N = rs.NumColumns
```

The next statement prints the value in N (the number of columns in rs) in the IDE Output panel.

```
Print "Number of columns = ", N
```

The next four lines are a For statement that iterates through the result set's columns. The number of columns was already stored in the variable N. The For statement starts at 1 and loops N times through the result set's columns. For each column, the script does the following:

- Returns the name of the column, using the FieldName method of the ResultSet object
- Stores the name in a variable called fName
- Prints the name in the IDE Output panel

After performing these actions for a column, the script iterates to the next column and repeats these steps until all columns have been processed. The following code performs these steps.

```
    For i = 1 to N
      fName = fs.FieldName (i)
      Print fName
    Next
```

If the query fails to run, the preceding steps (iterating through the table) are never performed and a message box displays the message “The query did not run successfully.” The MessageBox statement takes three arguments: the message to be displayed in the box, a value determining which buttons appear on the box (in this case a value of zero specifies a box with an OK button), and the message box name, in this case Query Execution.

```
  Else
    MessageBox "The query did not run successfully.",0,
"Query execution"
```

The last three lines of the script end the two If...Then...Else statements and deactivate the dBASE IV connection.

```
  End If
  con.Disconnect
End If
```

Chapter 6
OLE Automation Support

With OLE Automation you can access and control another application's objects. The application through which you manipulate such objects is called the automation controller. The application providing the objects you want to manipulate is called the automation server.

Using OLE Automation in Lotus Products

The following Lotus applications can function as both OLE Automation controllers and servers:

- Approach
- Freelance Graphics®
- Word Pro

Automation controllers

In addition to working with objects created using your current application, you can write scripts in LotusScript to control objects in applications other than the one in which you are running LotusScript. When you reference such an object in a script, the application in which you run the script is called the automation controller. For example, if you have Word Pro installed, you can write a script in Approach (the automation controller) that manipulates objects in Word Pro (the automation server).

Automation servers

When an automation controller accesses objects created by another application, the application in which the objects were created is called an automation server. The automation controller can manipulate or extract data from the automation server's objects by reading and writing properties and invoking methods. Every Lotus application that supports LotusScript can function as an OLE Automation server. You can access such OLE Automation servers using LotusScript, Visual Basic®, Visual C++™, or through other languages written for applications that support OLE Automation.

Accessing LotusObjects

Once you know how to write scripts that run independently within applications, you don't have to learn much more to take advantage of OLE Automation. When you write a script that controls another application's objects, you use mostly the same syntax that you would in a script written for that application.

The following script, written in LotusScript and running in Word Pro, is an example of an application manipulating its own objects. This script creates a new Word Pro document, inserts some text, and saves the file:

```
Dim app As Application

Set app = CurrentApplication
app.NewDocument ("Foo.Lwp")
app.Text.InsertText "This is a New Line"
app.Save
```

You can do the same thing from outside of Word Pro using an application that is running another scripting language. The next script is an example of an application manipulating objects in another application, using Visual Basic. The application running the Visual Basic script is the controller, and Word Pro is the server:

```
Dim app As Object

Set app = CreateObject ("WordPro.Application")
app.NewDocument ("Foo.Lwp")
app.Text.InsertText "This is a New Line"
app.Save
```

In this example, Visual Basic attaches to Word Pro and instructs Word Pro to create the new Word Pro document, insert some text, and save the file. The syntax of the Visual Basic example differs from the Word Pro example in the following two ways:

- The Visual Basic CreateObject function is used to access Word Pro via OLE Automation.
- The variable app is declared as an Object, rather than as an Application. Object is a Visual Basic data type.

For more information

For more information about OLE Automation in Word Pro, do the following.

1. In Word Pro, choose Help - Help Topics and click the Search button.
2. Type this phrase:

   OLE Automation
3. Click Show Topics, select a topic, and then click Go To.

LotusScript applications as controllers

Any product that supports LotusScript can act as an OLE Automation controller. The LotusScript functions CreateObject and GetObject allow you to write scripts that access OLE Automation objects. These objects are stored in variables of type Variant, a LotusScript data type. For information about LotusScript data types, see Chapter 3, “Data Types, Constants, and Variables,” in the *LotusScript Programmer’s Guide.*

The following script is an example of an application manipulating objects in another application, using LotusScript. The script runs in Approach or in any other application that supports LotusScript. Like the two preceding examples, it creates a new document, inserts some text, and saves the file. In this example Approach is an automation controller accessing Word Pro, an automation server:

```
Dim app As Variant

Set app = CreateObject ("WordPro.Application")
app.NewDocument ("Foo.Lwp")
app.Text.InsertText "This is a New Line"
app.Save
```

Notice that this example looks a lot like the Visual Basic example appearing in the preceding section. The only difference is that it declares the variable app as a Variant, rather than as an Object.

Object names for applications

An application object is at the start of the object hierarchy for its respective application. From an application object, you can traverse the hierarchy to find all other objects. To invoke the OLE Automation server for a Lotus application, use your scripting language's CreateObject method.

Lotus applications use the following object names when exposing their objects for OLE Automation:

- Approach.Application
- Freelance.Application
- WordPro.Application