LotusScript Help is not available

The LotusScript Help files are not installed on your system. You can reinstall Approach and specify that LotusScript Help should be installed.

1. Insert the CD-ROM in the appropriate drive and close the drive door.
2. Choose Run from the Start menu.
3. Type ***x*:install**, where *x:* is the appropriate drive.
4. Click OK. Follow the instructions in Install.
5. Choose "Customize features - Manual install" in the Install Options & Approach Folder dialog box.
6. Click Next.
7. Select "LotusScript Help for Approach" on the Approach tab.
8. Deselect all other features on all tabs.

You can obtain updates to these Help files through the Internet, CompuServe, or Lotus Customer Support.

Overview: Fulfillment Information

Refer to the following table when ordering the Application Developer's Documentation Set for SmartSuite 97.

- Identify the fulfillment center/centre for your country
- Copy the mailing address to the front of the fulfillment coupon
- Use the appropriate shipping and handling charge for your country

Currency	*Countries*	*Charge*	*Mailing address*
Australian Dollar	Australia	A$35	Lotus Development Pty Ltd Customer Service Department Level 12, 321 Kent Street Sydney NSW 2000 Australia
Canadian Dollar	Canada	C$15	Lotus Development Corporation SmartSuite Documentation P.O. Box 670 Scarborough, Ontario M1K 5C5
Belgian Franc French Franc Lira Guilder Escudo South African Rand Peseta	Belgium France Italy Netherlands Portugal South Africa Spain	900BF 150F L. 4500 F 50 4500 Esc. R135 3500Pts	Lotus Assistance SARL Parc Club Ariane Bat. Neptune 5 Bld des Chenes, BP 219 78051 St. Quentin en Yvelines Cedex FRANCE
Austrian Schilling Deutchmark Swiss Franc	Austria Germany Switzerland	300 OS 45 DM SFr 35	Lotus Development Gmbh Baierbrunnerstrasse 35 Postfach 70 12 20 81379 Muenchen GERMANY
New Zealand Dollar	New Zealand	NZ$40	Lotus Development New Zealand Ltd Customer Service Dept Level 20, ASB Bank Centre Cnr Albert & Wellesley Sts Auckland New Zealand
US Dollar	United States	$10	Lotus Development Corporation SmartSuite Documentation P.O. Box 25367 Rochester NY 14625-0367 USA
Danish Krone Markka Punt Norwegian Krone Krona Sterling	Denmark Finland Ireland Norway Sweden United Kingdom	Dkr 175 130 mk IR£15 Nkr 190 200 Skr £15	Lotus Development European Corporation Lotus Park The Causeway Staines Middlesex TW18 9AG ENGLAND
US Dollar	Others	US$30.00	Lotus Development Corporation SmartSuite Documentation P.O. Box 25367 Rochester NY 14625-0367 USA

Australia

Lotus Development Pty Ltd
Customer Service Department
Level 12, 321 Kent Street
Sydney NSW 2000
Australia

Canada

Lotus Development Corporation
SmartSuite Documentation
P.O. Box 670
Scarborough, Ontario M1K 5C5

France

Lotus Assistance SARL
Parc Club Ariane
Bat. Neptune 5
Bld des Chenes, BP 219

78051 St. Quentin en Yvelines Cedex
FRANCE

Germany
Lotus Development Gmbh
Baierbrunnerstrasse 35
Postfach 70 12 20
81379 Muenchen
GERMANY

New Zealand
Lotus Development New Zealand Ltd
Customer Service Dept
Level 20, ASB Bank Centre
Cnr Albert & Wellesley Sts
Auckland New Zealand

United Kingdom (U.K.)
Lotus Development European Corporation
Lotus Park
The Causeway
Staines Middlesex TW18 9AG
ENGLAND

United States
Lotus Development Corporation
SmartSuite Documentation
P.O. Box 25367
Rochester NY 14625-0367

Country	*Currency*	*SHCharge*	*Center*
Australia	Australian Dollar	A$30.00	Australia
Austria	Austrian Schilling	30.00 ÖS	Germany
Belgium	Belgian Franc	30.00 BF	France
Canada	Canadian Dollar	C$10.00	Canada
Denmark	Danish Krone	Dkr 30.00	U.K.
Eastern Europe	US Dollar	$30.00	
Finland	Markka	30.00 mk	U.K.
France	French Franc	30.00 F	France
Germany	Deutschmark	30.00 DM	Germany
Ireland	Punt	IR£30.00	U.K.
Italy	Lira	L. 30	France
Luxembourg	Luxembourg Franc	30.00 LF	France
Netherlands	Guilder	F 30.00	France
New Zealand	New Zealand Dollar	NZ$30.00	New Zealand
Norway	Norwegian Krone	Nkr 30.00	U.K.
Portugal	Escudo	30.00 Esc.	France
South Africa	South African Rand	R30.00	France

Spain	Peseta	30 Pts	France
Sweden	Krona	30.00 Skr	U.K.
Switzerland	Swiss Franc	SFr 30.00	Germany
United States	US Dollar	$10.00	United States
U.K.	Sterling	£30.00	U.K.

Developing SmartSuite Applications

Developing SmartSuite Applications Using LotusScript is available in the SmartSuite CD package as an online book. To install *Developing SmartSuite Applications Using LotusScript*, see the SmartSuite installation instructions.

To order a printed version of *Developing SmartSuite Applications Using LotusScript* and other LotusScript documentation, complete the order form and return it to Lotus.

Ordering the LotusScript Documentation Set

To order the *SmartSuite Application Developer's Kit,* complete the following form and mail or fax it to Lotus. You will receive the *SmartSuite Application Developer's Kit* within 21 days.

Ordering by US Mail

Mail the completed form to:

Lotus Development Corporation
55 Cambridge Parkway
Cambridge, MA 02142

Ordering by FAX

Fax the completed form to:

(617) 537-8500

Order Form

Name: _____

Title: _____

Mailing address: _____

_____

_____

_____

E-mail address: _____

Product Registration Number: _____

LotusScript Documentation as Online Books

You can use the SmartSuite Custom Install to install the following online books about LotusScript:

- *Getting the Most Out of LotusScript in SmartSuite 97*
- *Developing SmartSuite Applications Using LotusScript*
- *LotusScript Language Reference*
- *LotusScript Programmer's Guide*

For more information about installing online books, see your SmartSuite installation documentation.

LotusScript Documentation on the Web

You can view updated versions of LotusScript documentation, or download updated sample applications or Help files from the LotusScript home page.

Enter the following URL in the location field in your browser and press ENTER:

`www.lotus.com/home.nsf/welcome/sswin`

Script Design Basics

LotusScript provides a variety of tools and services to support you in developing applications for SmartSuite. Getting productive in a new programming environment often involves understanding how all the pieces work together -- the tools, the language conventions, the object dependencies, and so on. Understanding how to approach the problem and where to enter your script code is half the challenge in learning.

Choosing a place to begin

Lotus Notes, 1-2-3, Approach, Freelance Graphics, and Word Pro all use the same underlying LotusScript language. Each product implements LotusObjects on top of the LotusScript language. To determine what product best supports the goals for your script application, consider using each of the SmartSuite products and reviewing its features. Read *Developing SmartSuite Applications Using LotusScript* for overviews of what each product can bring to your programming effort. Implement a few simple procedures in each product to get a feel for its features and objects. In the long run, you'll be better able to determine what product provides strengths where you need them most, and how you can develop cross-product applications that take advantage of the strengths of each product.

Working the basics

LotusScript applications share the following common features.

- You need a Lotus product to run script applications.
- You need a Lotus product to store scripts in a product document such as a 1-2-3 workbook or Word Pro document.
- You need to run the Lotus Integrated Development Environment (IDE) to edit and debug scripts stored in a product document.
- You need to open an IDE window for each product document containing scripts that you want to modify.

To write a basic script application, therefore, you must run a Lotus product and load a document in that product. You can then write scripts for the product objects that you create in your product.

Writing scripts in the Integrated Development Environment (IDE)

Your primary tool for developing script applications is the Lotus Integrated Development Environment (IDE). Beyond providing the basic tools such as an editor, a debugger, a browser, and a dialog editor, the IDE provides a high degree of integration with each Lotus product. It is easy to move between tasks that you perform in a product and those that you perform in the IDE.

Writing global scripts

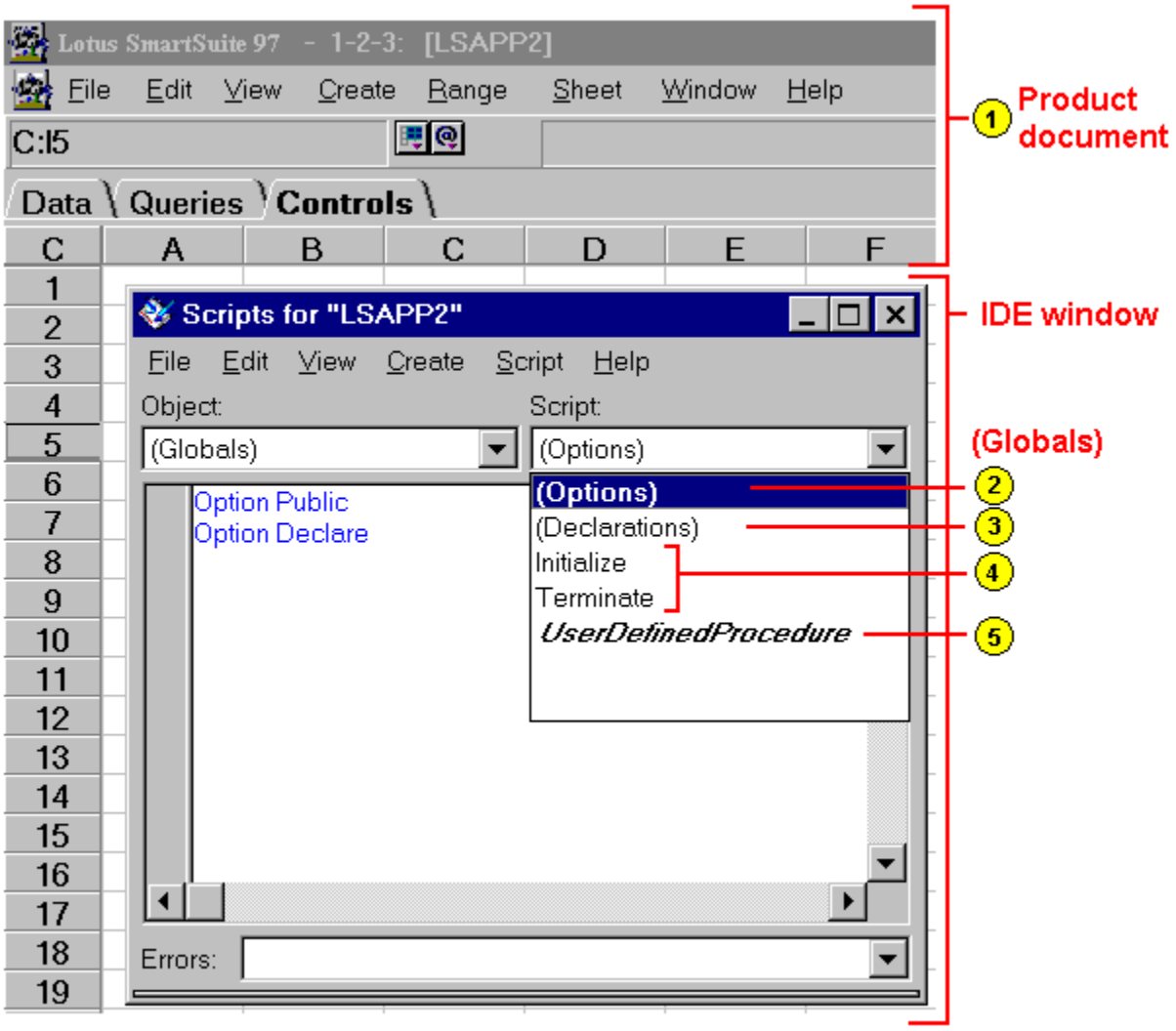
Global scripts make declarations, options, and procedures available to all scripts in your document. For example, to write global scripts for a 1-2-3 document named LSAPP2.123, you must first run 1-2-3, load the document LSAPP2.123, and then open an IDE window for that document. Choose Edit - Scripts & Macros - Show Script Editor in the 1-2-3 menu to activate an IDE window for your current document.

The IDE lists objects that you can script in the Object list and scripts for each of those objects in the Script list. You can add statements to predefined scripts in (Globals), such as (Options), (Declarations), Initialize, or Terminate, or you can create your own named procedures. You do not need to modify predefined scripts to write a basic script application.

The following illustration shows how to select a particular script for (Globals).

Click any item in the following list to learn more about it.

1. Product document
2. (Options) scripts
3. (Declarations) scripts
4. Initialize and Terminate subs
5. User-defined procedures



Writing scripts for product objects

You can also write scripts for product objects in your document. As with (Globals), you can add statements in the predefined scripts for an object, or create new procedures for that object. Unlike scripts that you write in (Globals), the declarations, options statements, and procedures that you write for a product object are not generally available to scripts attached to a different product object.

The predefined scripts for product objects include object event procedures. Script statements in an object event procedure are executed when an object, such as a button, receives a particular event in your product, such as being clicked, double-clicked, or moved. For example, if you add a button named Button 5 to the 1-2-3 document LSAPP2.123, and you want it to run some script when you click it, you must add script statements to the Click procedure for Button 5. To select this event procedure, choose the Button 5 object in the IDE Object list and choose Click in the Script list.

The following illustration shows how to select a predefined or user-defined script for a 1-2-3 product object named Button 5.

Click any item in the following list to learn more about it.

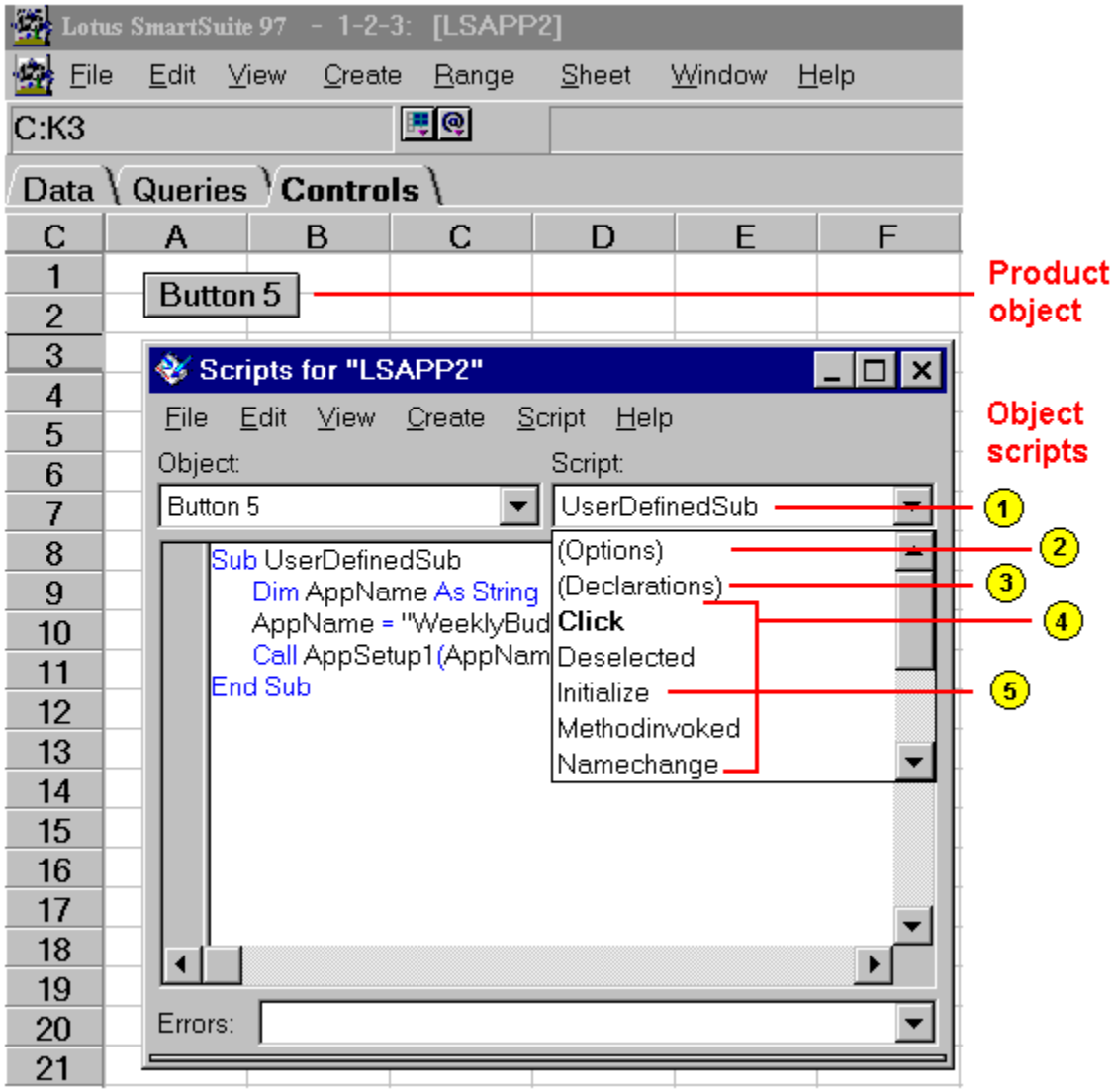
1 User-defined procedures

1 (Options) scripts

1 (Declarations) scripts

1 Event procedures

1 Initialize and Terminate subs



Working with external script files

In many cases, the one-application-per-document approach is sufficient for working with objects and data in isolated documents. To develop more sophisticated applications that reuse important scripts or use multiple products, consider using the following types of external script files:

LotusScript Script (LSS) files

LotusScript Object (LSO) files

LotusScript Extension (LSX) files

OLE Custom Control (OCX) files

Dynamic-link Library (DLL) files

Dynamic-link Library (DLL) files

If you develop useful functions in C and compiled them in a Dynamic-link Library (DLL), you can call them from your LotusScript application. For example, the following procedure declares and calls a LotusScript function named SendDLL, corresponding to a C function named _SendExportedRoutine in the DLL file named MYEXPORTS.DLL.

```
Declare Function SendDLL Lib _
   "C:\LOTUS\ADDINS\MYEXPORTS.DLL" _
   Alias "_SendExportedRoutine" (i1 As Long, i2 As Long)
SendDLL(5, 10)
```

For more information on using Dynamic-link Libraries, see *LotusScript Language Reference*.

(Declarations) scripts in (Globals)

The (Declarations) script is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts in your document
- Public, Private, Type, Class, and Declare Lib statements (external C calls)
- Const statements for those constants that you want to be available to all scripts in your document and are not needed for Use or UseLSX statements in (Options)

By default, the (Declarations) script is initially empty.

If you enter Type, Class, or Declare Lib statements in any other script in (Globals), the IDE moves them to (Declarations) automatically. If you enter Dim, Public, Private, or Const statements outside the scope of a procedure in another script, the IDE moves them to (Declarations) automatically. Const statements in (Options) are the exception to this rule.

Initialize and Terminate subs in (Globals)

Initialize script

Use the Initialize sub in (Globals) to initialize variables that you declared in (Declarations). The Initialize sub executes before any of these variables are accessed and before any other scripts in (Globals) are executed. By default, the Initialize script is empty.

Terminate script

Use the Terminate sub in (Globals) to clean up variables that you declared in (Declarations) when you close your document, or when you modify a script and execute it again. For example, you can use an Open statement to open a file containing data in Initialize, and use a Close statement in Terminate to close it. By default, the Terminate script is empty.

(Options) scripts in (Globals)

The (Options) script in (Globals) is designed to contain these the following statements:

- Option statements

  Note (Options) contains the statement, Option Public, by default. This makes Const, Dim, Type, Class, Sub, Function, and Property statements public by default. You can use the Public form of these statements to make them public explicitly, or the Private form to make them unavailable to other scripts outside (Globals).
- Def*type* statements
- Use and UseLSX statements
- Const statements needed for Use and UseLSX statements

If you enter any of these statements, except for Const, in any other script in (Globals), the IDE automatically moves them to (Options).

Option and Def*type* statements that you enter in (Options) apply only to scripts for the current object. To make certain that an option is applied consistently throughout your document, enter the appropriate statement in the (Options) script for every object for which you are writing scripts.

User-defined procedures in (Globals)

While you are working in (Globals), you can add procedures to make them available throughout your document. There are three ways to add procedures to (Globals) in the IDE:

- *Using the IDE menu*: Choose Create - New Sub or Create - New Function in the IDE menu to create new subs and functions in (Globals). The IDE automatically adds the name of the new procedure to the Script list.
- *Entering statements*: Enter a Sub, Function, or Property statement anywhere in (Globals) except within a class. The IDE automatically adds the name of the new procedure to the Script list for (Globals).
- *Importing procedures from a file*: Use File - Import Script in the IDE menu to import scripts when you are working in (Globals). These imported scripts will be available to all scripts in your document. The IDE automatically adds the name of any new procedures contained in the imported script to the Script list.

LotusScript User Assistance for SmartSuite

To help you learn how to develop LotusScript applications for SmartSuite, Lotus provides a complete library of user assistance.

Getting the Most Out of LotusScript in SmartSuite 97

This publication explains how SmartSuite 97 products use the LotusScript programming language and how your business can take advantage of LotusScript in developing applications for SmartSuite.

Getting the Most Out of LotusScript in SmartSuite 97 is available in hard copy, Adobe Acrobat, or HTML formats in your SmartSuite 97 package, in the SmartSuite Application Developer's Kit, or on the Worldwide Web.

Developing SmartSuite Applications Using LotusScript

This publication provides comprehensive information on key concepts and techniques for developing LotusScript applications. *Developing SmartSuite Applications Using LotusScript* focuses on programming tools, cross-application programming, Lotus Notes integration, and product-specific application development.

Developing SmartSuite Applications Using LotusScript is available in hard copy, Adobe Acrobat, or HTML formats in your SmartSuite 97 package, in the SmartSuite Application Developer's Kit, or on the Worldwide Web.

LotusScript Language Reference

This publication provides a comprehensive summary of conventions and basic commands for the LotusScript language. *LotusScript Language Reference* provides the foundation for programming any product that supports the LotusScript programming language.

LotusScript Language Reference is available in hard copy, Adobe Acrobat, Help, or HTML formats in your SmartSuite package, in the SmartSuite Application Developer's Kit, or on the Worldwide Web.

LotusScript Programmer's Guide

This publication is a general introduction to LotusScript that describes basic building blocks in the language and explains how to use them to create powerful applications.

LotusScript Programmer's Guide is available in hard copy, Adobe Acrobat, or HTML formats in your SmartSuite package, in the SmartSuite Application Developer's Kit, or on the Worldwide Web.

Class Reference Help and Frequently-asked Questions

Each product provides comprehensive Help on product classes, frequently-asked questions about programming, and code examples. All this is delivered in an innovative Help system designed to enhance your work as a programmer.

Class reference Help and frequently-asked questions are available in Help or HTML formats in your SmartSuite 97 package, in the SmartSuite Application Developer's Kit, or on the Worldwide Web.

Example code and sample applications

Most products also provide working code to illustrate important programming techniques. You can reuse and modify this code as you develop your own applications.

Example code is available in the SmartSuite CD-ROM package, in the SmartSuite Application Developer's Kit, and on the Worldwide Web.

LotusScript Object (LSO) files

LotusScript Object (LSO) files contain public definitions that you can use in your script applications. If you develop a library of commonly-used declarations or procedures that you want to reuse across multiple script applications, you can collect them in a product document, and use the File - Export Globals as LSO menu command to create a compiled LotusScript Object file. If this file were named WKREPORT.LSO, you would make these public definitions available to your script application by entering the following statement in the appropriate (Options) script:

```
Use "C:\LOTUS\ADDINS\WKREPORT.LSO"
```

For more information on using LotusScript Object files, see *LotusScript Language Reference*.

LotusScript Script (LSS) files

LotusScript Script (LSS) files are text files that contain LotusScript statements. You can create LSS files in any text editor. Use the %Include directive anywhere in a script to reference the contents of an LSS file. For example, to include the contents of a LotusScript Script file named STDSETUP.LSS in your application, enter the following statement:

```
%Include "C:\MYSCRIPTS\STDSETUP.LSS"
```

By default, LotusScript assumes that the LotusScript Script files referenced have an LSS file extension. You can actually use any extension for your text file or no extension at all.

For more information on using LotusScript Script files, see *LotusScript Language Reference*.

LotusScript Extension (LSX) files

LotusScript Extension (LSX) files are Dynamic-link Libraries (DLLs) that contain public class definitions. LSX files are developed using the Lotus LSX Toolkit. To obtain a version of the LSX Toolkit for your operating system, connect to the Lotus home page on the World Wide Web. Lotus ships LSX files for Lotus Notes and Approach; other LSX files are being developed for SmartSuite products by Lotus and by third-party developers. These extension files expand the range of classes that you can use in your LotusScript applications.

Tip You can enter a UseLSX statement in any script; the IDE automatically moves it to (Options).

Loading and using class definitions in LSX files

There are two ways to load and use the public class definitions in an LSX file.

- If the LSX file that you want to load is not registered in the Registry, you must refer to the LSX file directly in your UseLSX statement.

  ```
  UseLSX "C:\MYSCRIPTS\LSX4DB2.DLL"
  ```

- If an LSX is registered and you want to reference a class definition directly, you can enter the name of the class definition.

  ```
  UseLSX "ObjectName"
  ```

  In this example, LotusScript searches all entries under "LotusScriptExtensions" in the Registry for the specified class definition, and loads that definition.

  Note If the LSX file you want to load is registered in the Windows Registry, you can reference its Registry name and have Windows provide the appropriate DLL name and file path. SmartSuite 97 registers an LSX file that contains Notes public class definitions. To use these Notes class definitions in your cross-product script applications, enter the following statement:

  ```
  UseLSX "*Notes"
  ```

Viewing class definitions

Once you run a script containing a UseLSX statement and loaded an LSX file, you can browse its class definitions in the IDE Browser panel.

For more information on using LotusScript Extension files, see *LotusScript Language Reference*.

(Declaration) scripts in object scripts

The (Declarations) script for an object is designed to contain the following statements:

- Dim statements for variables that you want to be available to all scripts for the current object
- Const statements for those constants that you want to be available to all scripts for the current object and that are not needed for Use or UseLSX statements in (Options)

By default, the (Declarations) script is initially empty.

Event procedures in object scripts

If you are writing a script for an object, the Script list displays default event procedures for the selected object. In the IDE, you cannot create new event procedures for an existing product object because valid events for that object are defined by the product.

Initialize and Terminate subs in object scripts

Initialize sub

Use the Initialize sub to set up variables declared in the object's (Declarations) script. The Initialize sub for an object executes before any of its event procedures. By default, the Initialize script is empty.

Note Scripts for controls created in the Lotus Dialog Editor do not have Initialize subs.

Terminate sub

Use the Terminate sub to clean up variables that you declared in the object's (Declarations) script. By default, the Terminate script is empty.

Note Scripts for controls created in the Lotus Dialog Editor do not have Terminate subs.

(Options) scripts in object scripts

The (Options) script for an object is designed to contain these the following statements:

- Option statements
- Def*type* statements
- Use and UseLSX statements
- Const statements needed for Use and UseLSX statements

User-defined procedures in object scripts

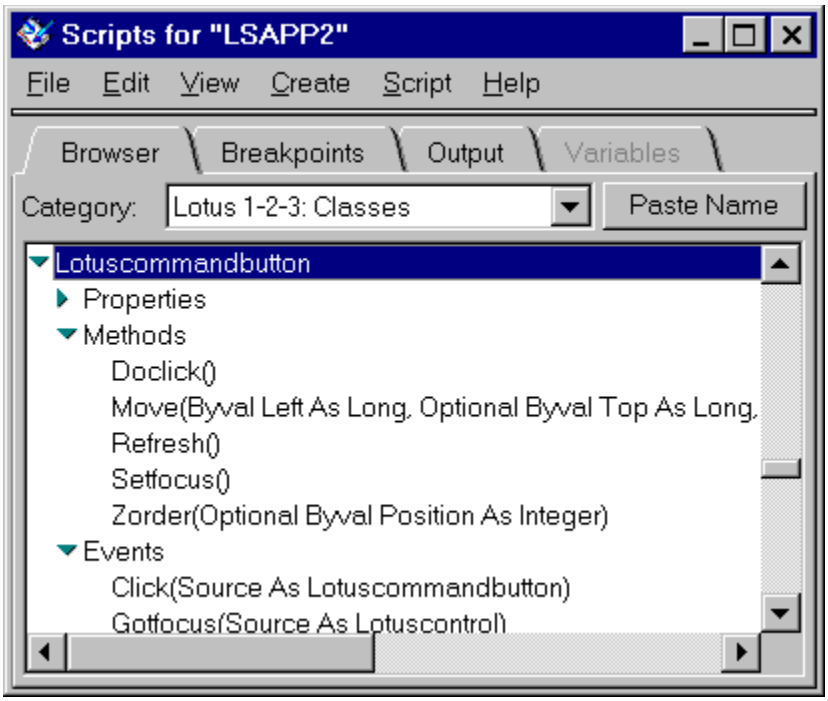
You can create other named subs, functions, and properties for objects, in addition to the predefined scripts or event procedures. Because these procedures are not in (Globals), they can be called only from other scripts for the object.

There are three ways to create object scripts in the IDE:

- *Using the IDE menu*: Use Create - New Sub and Create - New Function to create new subs and functions for an object. The IDE automatically adds the name of the new procedure to the Script list for that object.
- *Entering statements*: Enter a Sub, Function, or Property statement anywhere in a script for the current object. The IDE automatically adds the name of the new procedure to the Script list for that object.
- *Importing procedures from a file*: Use File - Import Script when you are working with object scripts to import scripts for that object. The IDE automatically adds the name of any new procedures contained in the imported script to the Script list.

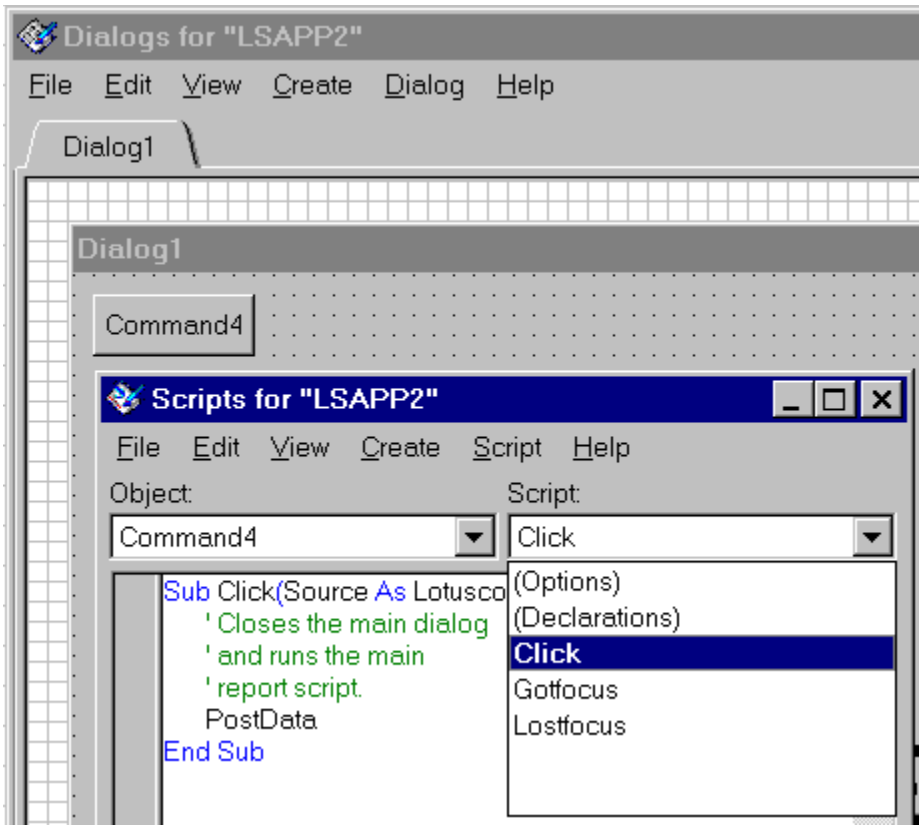
OLE Custom Control (OCX) files

OLE Custom Controls extend the number of objects that you can script in Lotus products. For example, the Lotus dialog controls listed under product classes in the IDE Browser panel are OCX controls that you can add to the Lotus Dialog Editor.



Once you add an OCX control to your product, you can script its properties, methods, and events in the IDE Script Editor.

The following illustration shows how the properties, methods, and events of an Lotus CommandButton OCX named Command4 are available to you in the IDE.



Tip You can add OCX controls registered on your system to the Lotus Dialog Editor Toolbox by choosing File - Toolbox Setup in the Lotus Dialog Editor menu.

Product Document

To edit scripts in the IDE or to execute them in one or more products, you must create or use a document in your product that contains the scripts. Lotus products supporting LotusScript use the following document extensions:

Lotus Product	*Document extension(s)*
1-2-3	123
Approach	APR
Freelance Graphics	PRZ SMC
Notes	NSF
Word Pro	LWP

Using LotusScript Examples

Code examples provide working models for the scripts that you write. Whether the example is listed in a Help example or available as a product document on disk, you can copy statements or entire scripts from the examples and use them in your own script applications.

There are three types of LotusScript examples, each designed to illustrate a different aspect of the LotusScript language or the classes available for each SmartSuite product.

Examples in reference Help

Most examples appear in reference Help for the LotusScript language and for product classes. These brief examples focus on individual elements in the language or members of a product class. They illustrate how to use correct syntax for a working example, how to enter appropriate values for parameters, and how dependencies between elements operate.

Note Although you can copy examples from reference Help and paste them into your scripts, they are not designed primarily to be self-contained. Sometimes there are dependencies between a piece of example code and the larger sample application from which it is derived.

Examples in Frequently Aasked Questions (FAQs) Help

Frequently Asked questions (FAQs) illustrate how to complete common programming tasks using LotusScript. Examples in FAQs not only illustrate how individual statements work, but they also illustrate how these statements form a complete application or procedure. Most examples in FAQs are designed to be self-sufficient; you can copy one or more procedures from Help, paste them into your own scripts in the Script Editor, and execute them.

Note When there are dependencies in an example that would require you to modify the example to make it run, these dependencies are documented in the Help topic or at the beginning of the first script in the example.

Sample applications

The *Developing SmartSuite Applications Using LotusScript* book includes numerous sample applications for SmartSuite and for individual products. These examples are designed to illustrate more sophisticated tasks for an individual product or tasks that utilize more than one product. They illustrate how to develop script applications that take advantage of embedded OLE objects, OLE automation, Notes, Visual Basic, the Worldwide Web, and custom Dynamic-link libraries (DLLs). Lotus develops new sample applications for SmartSuite on an ongoing basis; these new samples and updated versions of the ones in *Developing SmartSuite Applications Using LotusScript* are available on the World Wide Web.

To copy scripts from these sample applications and paste them into your own script applications, you must first open the sample application document and then display its scripts by opening the IDE window for that document.

Note All sample applications in *Developing SmartSuite Applications Using LotusScript* are designed to run without modification.

Using LotusScript Help

The design for LotusScript Help supports three of the most frequent activities that you perform as a programmer:

- Searching for objects and elements to use in your scripts
- Writing scripts
- Debugging scripts

LotusScript Help uses different types of windows to display different types of information, so it is important to know what each type of window contains and how to navigate between them.

Using Help to search for objects and elements

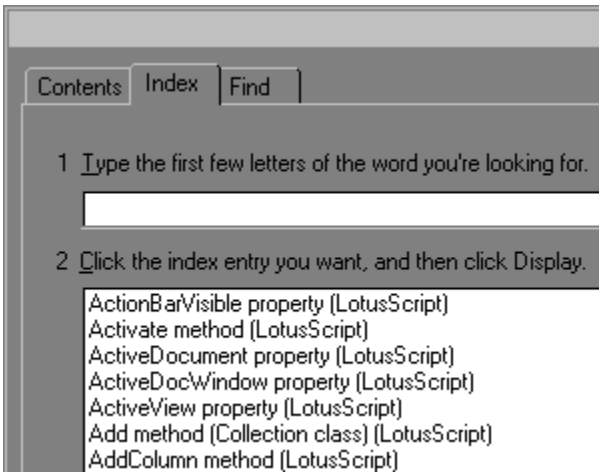
There are areas in Help designed to help you search for objects and language elements to use in your scripts:

LotusScript Help Contents

You can use Contents in Help to examine the overall structure of Help and to browse for Help topics relevant to your current script.

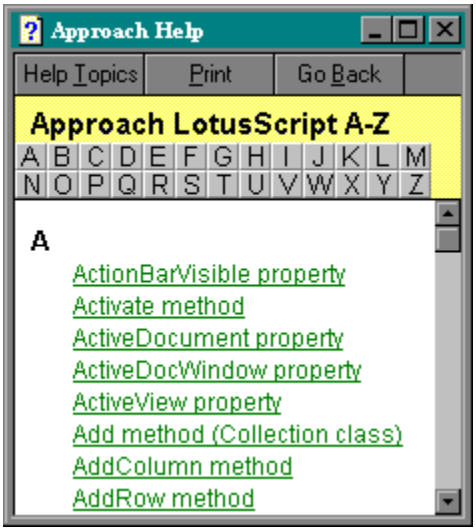
LotusScript Index

Indexes are one of the most popular ways that programmers search for information. Topics in LotusScript Help are indexed alphabetically, so you can enter key phrases or keywords and navigate to the corresponding Help topics.



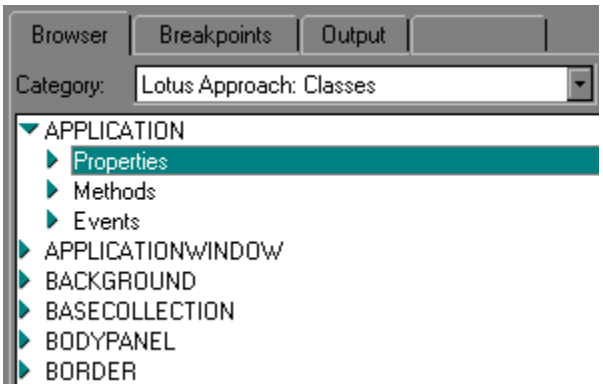
LotusScript A - Z lists

LotusScript Help for each product provides A - Z lists of its classes, properties, methods, and events, including a comprehensive list of all the elements in the product.



IDE Browser Help

The Browser panel in the Integrated Development Environment (IDE) displays lists of LotusScript language elements and classes for products. You can expand and collapse entries in the Browser to view the associated properties, methods, and events for objects.



Highlight an element in the Browser panel and press F1 (HELP) to get context-sensitive Help on that element.

Using Help to write scripts

Help focuses on objects. As you are writing scripts, you explore the relationships between product classes and the behaviors of objects in that product.

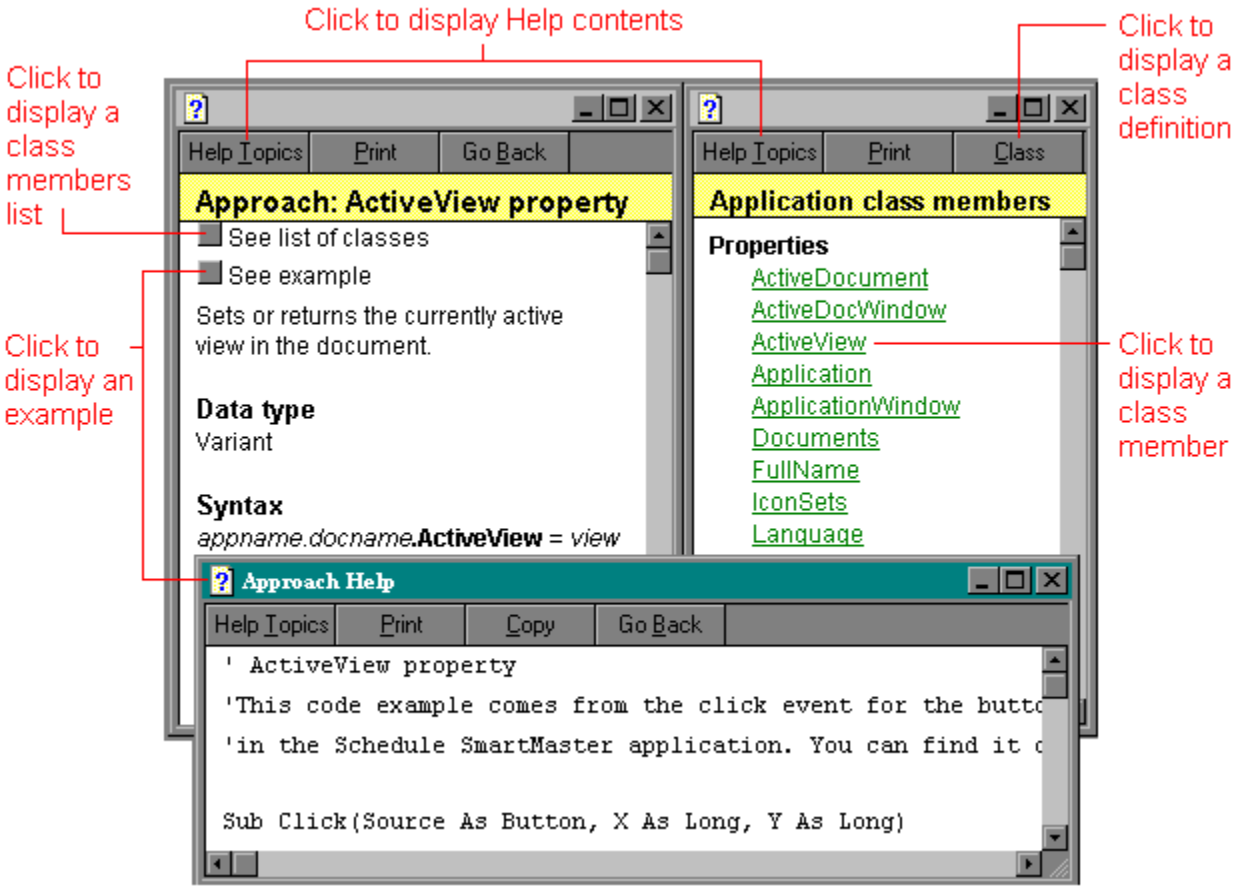
Types of Help windows

To support this exploration, Help separates information about classes into four types of windows:

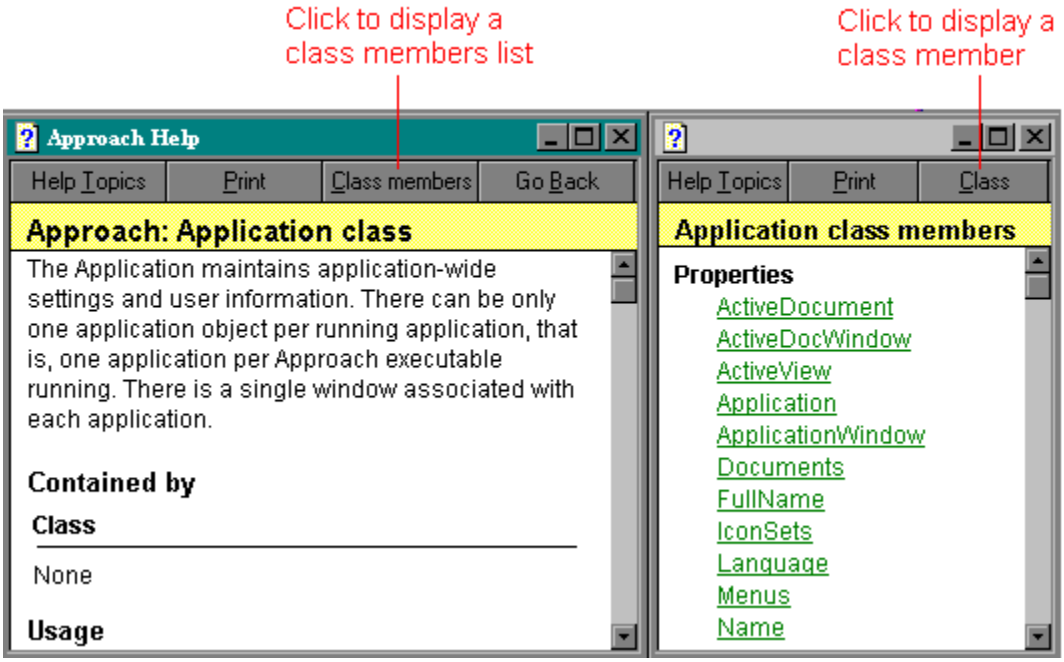
- Class definition windows define what a class does in a product and how it works in the product's containment hierarchy. The class definition topic for the 1-2-3 Range object describes what ranges do in 1-2-3, how they are contained by larger objects, and how they contain smaller objects.
- Class member list windows list all the properties, methods, and events that are members of a particular class.
- Class member windows focus on particular properties, methods, or events.
- Example windows contain one or more scripts for a particular property, method, or event. You can copy and paste script statements from these example windows into the IDE Script Editor.

Displaying Help windows

To display different types of LotusScript Help windows, use buttons in Help topics and in the Help window that are labeled by the type of Help window. The following illustration shows how to use buttons to display class member, class member list, and example windows in Help.



The following illustration shows how to display class definition and class member list windows in Help.



Help for editing and debugging scripts

You can also get context-sensitive Help about keywords and messages when you are editing or debugging your

scripts in the IDE.

Context-sensitive Help in the Script Editor and Script Debugger

If you need help on a keyword while you are writing or debugging a script in the Script Editor and Script Debugger, place the insertion point on the keyword and press F1 (HELP) to get context-sensitive Help on that keyword.

Context-sensitive Help on messages

You can also get context-sensitive Help on two types of messages in the IDE. In the Script Editor, you can get context-sensitive Help on syntax errors. Navigate to the statement that caused the error and press F1 (HELP). When you are debugging your scripts and the IDE reports a run-time error, press F1 (HELP) to display information about that error and suggestions about fixing it.

```
' ActionBarVisible property
Sub CleanScreen()
     ' This program performs the same function
     ' that the Clean Screen menu item on the Edit
     ' menu does.
     CurrentWindow.Redraw=False   ' Turn off redraw temporarily
     ' Turn off each bar.
     CurrentWindow.ActionBarVisible=False
     CurrentWindow.IconBarVisible=False
     CurrentWindow.StatusBarVisible=False
     CurrentWindow.ViewTabVisible=False
     CurrentWindow.Redraw=True   ' Turn it redraw back on
     CurrentWindow.Repaint       ' Now repaint the window.
End Sub
```

```
' Activate method
' Activate makes the specified document the active document and brings
' it to the foregound. In this example, we have 2 documents (.APR's),
' Customer and Accounts. The user is currently looking at the Customer
' document, that is, the Customer document is active. This simple global
' function is passed a document. The function will activate the document
' that is passed to it.

Sub MakeDocActive(Doc As Document)
    Application.Doc.Activate
End Sub
```

```
' ActiveDocument property
' This script checks to see if the active document is the Customer document.
' If it isn't, it makes the Customer document active.

' Check if the active document is Customer.
If (CurrentApplication.ActiveDocument.Name <> "Customer") Then
    ' Make the Customer document active.
    CurrentApplication.Customer.Activate
End If
```

```
' ActiveDocWindow property
' Retrieve the active document window.

Dim DocWin As DocWindow
Set DocWin = CurrentApplication.ActiveDocWindow
```

```
' ActiveView property
' This code example comes from the click event for the button btnToday
' in the Scheduler SmartMaster application. You can find it in the Start view.

Sub Click(Source As Button, X As Long, Y As Long)
     Set CurrentWindow.ActiveView = CurrentDocument.Schedule~ Display
     CurrentView.Body.fbxDateDisplay.Text = Format$(Now, "m/d/yy")
     clearDisplay
     readBlock CurrentView.Body.fbxDateDisplay.Text
End Sub
```

```
' AddColumn method
' Example 1
' Create a new worksheet and use the AddColumn method to add the
' field QTY to the new worksheet. Arguments to name the column and
' position the column are optional.
' Context for this example is a click event of a button
' on a form in the current document.

Dim Wrk As Worksheet
Set Wrk = New Worksheet(currendocument)
Set CurrentWindow.ActiveView=Wrk
Wrk.Name="Wrksheet"
Wrk.AddColumn("QTY")

' Example 2
' Add a new column to the current worksheet just to the left of the
' worksheet column named QTY2. Name the new column MyCol.
' The new column shows the values for the QTY field.

Dim AddCol As Integer
AddCol=CurrentView.AddColumn("QTY","MyCol","QTY2")
```

AddRow method

```
'This is the modifyRooms global function from the Schedule
'SmartMaster application.

Function ModifyRooms
     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 = CurrentDocument.Tables(0).Path & "rooms.dbf"
          Set RS.Query = Qry
          If (RS.Execute)Then
               If (RS.NumRows) Then
                    RS.FirstRow
                    Do
                         RS.DeleteRow
                    Loop While (RS.NumRows)
               End If
               ModifyRooms = True
               For i = 0 To Ubound(rooms)
                    RS.AddRow
                    RS.SetValue "room", rooms(i)
                    RS.UpdateRow
               Next
          End If
     End If
     Con.Disconnect
End Function
```

```
' Add method
' Create and run a find for distinct records without other conditions.
Sub DuplicateRecords
   'Create a FindDuplicate object.
   Set DupFind = New FindDuplicate("LastName")

   'Add another field to the FindDuplicate object definition.
   'Records must have the same value in both fields.
   Call DupFind.Add("FirstName")

   'Leave the first duplicated record out of the found set.
   Dim SuccessFlag As Integer
   SuccessFlag = DupFind.ExcludeFirst

   'Run the find.
   Call CurrentWindow.FindSort(DupFind)
End Sub

' Create a FindDuplicate object as part of a Find object.
Sub FindDups
   'Create a FindDuplicate object.
   Set DupFind = New FindDuplicate("LastName")

   'Add another field to the FindDuplicate object definition.
   'Records must have the same value in both fields.
   Call DupFind.Add("FirstName")

   'Leave the first duplicated record out of the found set.
   Dim SuccessFlag As Integer
   SuccessFlag = DupFind.ExcludeFirst

   'Create a Find object.
   Set MyFind = New Find()

   'Add a condition to find all records in Japan.
   Call MyFind.And("Country", "Japan")

   'Attach the FindDistinct object to the Find object.
   Set MyFind.FindSpecial = DupFind

   'Run the find.
   Call CurrentWindow.FindSort(MyFind)

End Sub
```

```
' Add method (Sort class)
' Create and run a sort using values from more than one field.
Sub SampleSort
   'This sub expects the main table for the current view to have the following fields:
   '   Country      Text
   '   OrderTotal   Numeric

   'This script sorts records according to the values in two different fields.
   'If finds all records, then sorts them in ascending order alphabetically by the
values
   'in the Country field, and then sorts them in descending order numerically by the
   'values in the OrderTotal field.
   Call CurrentWindow.FindAll
   Dim ssortAll As New sort
   Call ssortAll.Add("Country", LtsSortAscending)
   Call ssortAll.Add("OrderTotal", LtsSortDescending)
   Call CurrentWindow.FindSort(ssortAll)

End Sub
```

```
' Alignment property
' Right-align the data in the current object.
Source.Alignment = $LtsAlignmentRight

' Or

' Retrieve the alignment of the data in the current object.
Dim align As Integer
align = Source.Alignment
```

```
' AllowDrawing property
' PicturePlus fields do not allow drawing by default,
' but you can allow drawing in a PicturePlus field.
' ObjPictPlus is the name of the PicturePlus field.

source.ObjPictPlus.AllowDrawing = True
```

```
' And method

Sub SampleFinds
   'This sub expects the main table for the current view to have the following fields:
   '   Country      Text
   '   OrderTotal   Numeric

   Call CurrentWindow.FindAll

   'This script shows a correct way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneField As New Find
   Call sfindOneField.And("Country","Japan, USA")
   Call CurrentWindow.FindSort(sfindOneField)

   'This script shows an alternate way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneFieldAlternate As New Find
   Call sfindOneFieldAlternate.And("Country","Japan")
   Call sfindOneFieldAlternate.Or("Country","USA")
   Call CurrentWindow.FindSort(sfindOneFieldAlternate)

   'This script finds multiple values in multiple fields.
   'It finds records with either "Japan" or "USA" in the Country field with Order
Total >= 1000
   'and it also finds records with only "Japan" in the Country field with Order Total
<1000.
   'The two sets of results are returned as the found set.
   Dim sfindComplex As New Find
   Call sfindComplex.And("Country", "Japan, USA")
   Call sfindComplex.And("OrderTotal",">= 1000")
   Call sfindComplex.Or("Country", "Japan")
   Call sfindComplex.And("OrderTotal","<1000")
   Call CurrentWindow.FindSort(sfindComplex)

   'This script shows an INCORRECT way to finding multiple values in the same field.
   'Do not do this by mistake.
   Dim sfindBad As New Find
   Call sfindBad.And("Country","Japan")
   Call sfindBad.And("Country","USA")      'The value "USA" replaces the previous
value.
   Call CurrentWindow.FindSort(sfindBad)
   'The find returns only records with Country = "USA".

   ' This script sorts records according to the values in two different fields.
   'If finds all records, then sorts them in ascending order alphabetically by the
values
   'in the Country field, and then sorts them in descending order numerically by the
```

```
    'values in the OrderTotal field.
    Call CurrentWindow.FindAll
    Dim ssortAll As New sort
    Call ssortAll.Add("Country", LtsSortAscending)
    Call ssortAll.Add("OrderTotal", LtsSortDescending)
    Call CurrentWindow.FindSort(ssortAll)

End Sub
```

```
' ApplicationWindow property
' Retrieve the ApplicationWindow object.

Dim AppWin as ApplicationWindow
Set AppWin = CurrentApplication.ApplicationWindow
```

```
' Application property
' Retrieve the Application object.

Dim App As application
Set App = CurrentApplication.Application
```

```
' Author property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    ' If the document has been modified...
    If (CurrentDocument.Modified) Then
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' Background property
' Set the background property of a panel to red.
Source.Background.Color.SetRGB(COLOR_RED)

' Set the background of panel1 to be the same as panel2.
Set Source.panel1.Background = Source.panel2.Background
```

```
' Baseline property
' Display a dashed line in the current field box.

Source.Border.Baseline = True
```

```
' Black property
' Find out how much black is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Black   ' Determine the amount of black.
Print b                             ' Print the amount of black.
```

```
' Blue property
' Find out how much blue is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Blue    ' Determine the amount of blue.
Print b                             ' Print the amount of blue.
```

```
' Border property
' Change the border of the current display element to blue.
Source.Border.Color.SetRGB(COLOR_BLUE)
```

```
' Bottom (Border) property
' This script comes from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application.
' Create a new text block with a 1 point left and right ultramarine border.

Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    t = 1635 & (330 * t)
    Set tt = New textbox(currentview.Body)
    tt.text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $ItsBorderThick
    tt.Border.Color.SetRGB (color_ultramarine)
End Sub
```

```
' BringToFront method
' MyCompany is a field box on a form.
' Place the field box on top of any other display element that
' is in the same area of the form.
Source.MyCompany.BringToFront
```

Cascade method

```
' Cascade all of the open documents (.APR files).

CurrentApplication.ApplicationWindow.Cascade
```

```
' CheckedValue property
' Determine if a check box is checked, and then display a message
' that includes the checked or unchecked value.
' This script is placed in an event script for an object in the same
' view as the check box.
If (Source.ObjCheckBox.IsChecked) Then
    MessageBox("The check box is checked and its value is " &
Source.ObjCheckBox.CheckedValue)
Else
    MessageBox("The check box is not checked and its value is " &
Source.ObjCheckBox.UnCheckedValue)
End If
```

```
' ClickedValue property
' This script is called from the event of an object
' other than the RadioButton object.
' ObjRadio is the name of the radio button object.

Source.ObjRadio.ClickedValue = "VISA"

' Or

' Get the ClickedValue of the radio button.

Dim Val As String
Val = Source.ObjRadio.ClickedValue
```

```
' CloseWindow method
' This function lays the groundwork for an application that runs
' in the background. After opening a document, you can hide Approach.

Dim rval As Integer   ' Return Value

' Minimize Approach (optional)
CurrentApplication.ApplicationWindow.Minimize

' Hide Approach from the user
CurrentApplication.Visible=False

' Have Approach open an application that runs automatically.
rval = CurrentApplication.OpenDocument("AutoApp", "C:\LOTUS\APPROACH")

' Run your program.

' Show Approach to the user
CurrentApplication.Visible=True

' Maximize Approach (optional)
'CurrentApplication.ApplicationWindow.Maximize

' Quit Approach
CurrentApplication.CloseWindow
```

```
' Close method
' Determine the contents of the date field, then close the current window.
Sub Click(Source As Button, X As Long, Y As Long)
    Dim d As String

    d = Source.fbxDate.Text
    CurrentWindow.Close()
End Sub

' This script closes the view when users click the Cancel button.
Sub Click(Source As Button, X As Long, Y As Long, Flags As Long)
' Click event for the btnCancel button object on the Enter Date view
    CurrentWindow.Close
End Sub
```

```
' Close (ResultSet)
'This code is pulled from the 'deleteScheduledRemovedRooms' global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
     Dim C As New Connection
     Dim Q As New Query
     Dim RS As New ResultSet

     Dim tname As String
     tname = currentdocument.tables(0).tablename

     If C.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Q.Connection = C
          Q.Tablename = currentdocument.tables(0).path & tname
               Set RS.Query = Qry
               If (RS.Execute)Then
                    If (RS.numrows) Then
                         RS.firstrow
                         Do
                              RS.deleteRow
                         Loop While (RS.numrows)
                    End If
               End If
               RS.Close
          c.disconnect
     End If
End Sub
```

```
' Connection property
'This code is pulled from the 'deleteScheduledRemovedRooms' global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
     Dim Con As New Connection
     Dim Qry As New Query
     Dim ResSet As New ResultSet

     Dim TName As String
     TName = CurrentDocument.Tables(0).TableName

     If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Qry.Connection = Con
          Qry.TableName = CurrentDocument.Tables(0).Path & TName
               Set ResSet.Query = Qry
               If (ResSet.Execute)Then
                    If (ResSet.NumRows) Then
                         ResSet.FirstRow
                         Do
                              ResSet.DeleteRow
                         Loop While (ResSet.NumRows)
                    End If
               End If
          Con.Disconnect
     End If
```

```
'ConnectTo method
'Examples are provided for connecting from Approach to dBASE IV,
'DB2, SQL, Informix 5, Informix 7, Oracle 7, and Sybase 10 and 11.

'This code is from the 'deleteScheduledRemovedRooms' global
'function in the Meeting Room Scheduler SmartMaster application.
'It deletes all of the rows in the specified table.

Sub deleteScheduledRemovedRooms
   'Initialize variables for the connection.
   Dim Con As New Connection()
   Dim Qry As New Query()
   Dim ResSet As New ResultSet()
   'Create a shorter way to identify the table.
   Dim TName As Table
   'TName identifies the first (counting from zero) table attached
   'to this .APR file.
   Set TName = CurrentDocument.Tables(0)

   If Con.ConnectTo("dBASE IV") Then   'If the connection is successful, then.
      'Associate this Connection object with the new Query.
      Set Qry.Connection = Con
      Qry.TableName = TName.FullName
      'Associate this Query object with the new ResultSet.
      Set ResSet.Query = Qry
         If (ResSet.Execute)Then      'If the ResultSet is successful, then.
            If (ResSet.NumRows) Then   'If the ResultSet isn't empty, then.
               ResSet.FirstRow
               'Delete the rows in the table.
               Do
                  ResSet.DeleteRow
               Loop   While (ResSet.NumRows)
            End If
         End If
      'Close the connection.
      Con.Disconnect
   End If
End Sub
'This example describes two different methods for
'connecting to IBM DB2.

    'The first method for connecting to DB2:

    Dim Con As New Connection
    Dim Qry As New Query
    Dim RS As New Resultset
```

```
    Dim strUserid, strPassword, strDSname As String
    Dim strSQL, strSelect, strFrom, strWhere As String

'The second method requires additional connections,
'table names, and categories.

    'Define  the connection.
    strDSname = "SAMPLE"
    strUserid = "USERID"
    strPassword = "password"

    'Define the SQL statement.
    strSelect = "Select * "
    'The table name is case-sensitive in DB2.
    'The format is Owner.Tablename.

    'If the table name is in lowercase,
    'the From clause should be defined as:
    'strFrom = "From USERID.""employee"" "

    strFrom = "From USERID.EMPLOYEE "

    'If the field name is in lowercase,
    'the WHERE clause should be defined as:
    'strWhere = "Where ""firstname"" = 'MAUDE' or ""salary"" < 25000 "

    strWhere = "Where FIRSTNME='MAUDE'  or SALARY < 25000 "

    strSQL = strSelect & strFrom & strWhere
    If Con.Connectto("IBM DB2", strUserid, strPassword, strDSname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

    'The second method for connecting to DB2:
```

```
    If Con.Connectto("ODBC Data Sources", strUserid, strPassword, "!" &
strDSname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

End Sub

'This example describes two different methods for
'connecting to Informix5 SE Server.

    'The first method for connecting to Informix5 SE Server:

    Dim Con As New Connection
    Dim Qry As New Query
    Dim RS As New Resultset
    Dim strUserid, strPassword, strSrvName, strDBname, strDSname As String
    Dim strSQL, strSelect, strFrom, strWhere As String

'The second method requires additional connections,
'table names, and categories.

    'Define  the connection.
    strSrvName = "inf5_srv"
    strDBname = "sample"
    strDSname = "inf5"
    strUserid = "userid"
    strPassword = "password"

    'Define the SQL statement.
    strSelect = "Select * "

    'The table name is converted to lowercase in Informix.

    strFrom = "From janny1.employee "
```

```
    'The field name is converted to lowercase in Informix.

    strWhere = "Where firstnme='MAUDE'  or salary < 25000 "

    strSQL = strSelect & strFrom & strWhere
    If Con.Connectto("Informix 5", strUserid, strPassword, strSrvName  & "!!
DB=" & strDBname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

    'The second method for connecting to Informix5 SE Server:

    If Con.Connectto("ODBC Data Sources", strUserid, strPassword, strSrvName &
"!!DB=" & strDBname & ";DSN=" & strDSname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

End Sub

'This example describes two different methods for
'connecting to Informix7 WorkGroup Server.

    'The first method for connecting to Informix7 WorkGroup Server:

    Dim Con As New Connection
    Dim Qry As New Query
```

```
    Dim RS As New Resultset
    Dim strUserid, strPassword, strSrvName, strDBname, strDSname As String
    Dim strSQL, strSelect, strFrom, strWhere As String

'The second method requires additional connections,
'table names, and categories.

    'Define  the connection.
    strSrvName = "inf7_srv"
    strDBname = "sample"
    strDSname = "inf7"
    strUserid = "userid"
    strPassword = "password"

    'Define the SQL statement.
    strSelect = "Select * "

    'The table name is converted to lowercase in Informix.

    strFrom = "From sample:janny1.employee "

    'The field name is converted to lowercase in Informix.

    strWhere = "Where firstnme='MAUDE'  or salary < 25000 "

    strSQL = strSelect & strFrom & strWhere
    If Con.Connectto("Informix 7", strUserid, strPassword, strSrvName  & "!!
DB=" & strDBname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

      'The second method for connecting to Informix7 WorkGroup Server:

    If Con.Connectto("ODBC Data Sources", strUserid, strPassword, strSrvName &
"!!DB=" & strDBname & ";DSN=" & strDSname) Then
        Qry.SQL = strSQL
```

```
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

End Sub

'This example describes two different methods for
'connecting to Oracle Server.

    'The first method for connecting to Oracle Server:

    Dim Con As New Connection
    Dim Qry As New Query
    Dim RS As New Resultset
    Dim strUserid, strPassword, strSrvName, strDSname As String
    Dim strSQL, strSelect, strFrom, strWhere As String

'The second method requires additional connections,
'table names, and categories.

    'Define the connection.
    strSrvName = "orasvr"
    strDSname = "Oracle7 tables-Approach97"
    strUserid = "USERID"
    strPassword = "password"

    'Define the SQL statement.
    strSelect = "Select * "

    'To maintain case-sensitivity of the table name
    'or field name when using Oracle, place double
    'quotes around the table name or field name.

    'If the table name is in lower case, the From
    'clause should be defined as:
    'strFrom = "From USERID.""employee"" "
```

```
    strFrom = "From USERID.EMPLOYEE "

    'If the field name is in lower case, the WHERE
    'clause should be defined as:
    'strWhere = "Where ""firstname"" = 'MAUDE' or ""salary"" < 25000 "

    strWhere = "Where FIRSTNME='MAUDE'  or SALARY < 25000 "

    strSQL = strSelect & strFrom & strWhere
    If Con.Connectto("SQL Server", strUserid, strPassword, strSrvName) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

      'The second method for connecting to Oracle Server:

    If Con.Connectto("ODBC Data Sources", strUserid, strPassword, strSrvName &
"!" & strDSname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

End Sub

'This example describes two different methods for
'connecting to Microsoft/Sybase SQL Server.

    'The first method for connecting to SQL Server:
```

```
    Dim Con As New Connection
    Dim Qry As New Query
    Dim RS As New Resultset
    Dim strUserid, strPassword, strSrvName, strDBname, strDSname As String
    Dim strSQL, strSelect, strFrom, strWhere As String

'The second method requires additional connections,
'table names, and categories.

    'Define  the connection.
    strSrvName = "sqlsvr"
    strDBname = "testing"
    strDSname = "MS Sybase SQL Server-Approach97"
    strUserid = "userid"
    strPassword = "password"

    'Define the SQL statement.
    strSelect = "Select * "
    'The table name is case-sensitive in SQL Server.
    'The format is Database.Owner.Tablename.

    'If the table name is in lower case, the From
    'clause should be defined as:
    'strFrom = "From testing.userid.employee "

    strFrom = "From testing.userid.EMPLOYEE "

    'If the field name is in lower case, the WHERE
    'clause should be defined as:
    'strWhere = "Where firstname = 'MAUDE' or salary < 25000 "

    strWhere = "Where FIRSTNME='MAUDE'  or SALARY < 25000 "

    strSQL = strSelect & strFrom & strWhere
    If Con.Connectto("SQL Server", strUserid, strPassword, strSrvName) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
```

```
        End If
    End If
    Con.Disconnect

    'The second method for connecting to SQL Server:

    If Con.Connectto("ODBC Data Sources", strUserid, strPassword, strSrvName &
"!" & strDSname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

End Sub

'This example describes two different methods for
'connecting to Sybase System 10 Server.

    'The first method for connecting to Sybase System 10 Server:

    Dim Con As New Connection
    Dim Qry As New Query
    Dim RS As New Resultset
    Dim strUserid, strPassword, strSrvName, strDBname, strDSname As String
    Dim strSQL, strSelect, strFrom, strWhere As String

'The second method requires additional connections,
'table names, and categories.

    'Define  the connection.
    strSrvName = "sybsvr"
    strDBname = "testing"
    strDSname = "SYBNT"
    strUserid = "userid"
    strPassword = "password"

    'Define the SQL statement.
```

```
    strSelect = "Select * "
    'The table name is case-sensitive in SQL Server.
    'The format is Database.Owner.Tablename.

    'If the table name is in lowercase, the From
    'clause should be defined as:
    'strFrom = "From testing.userid.employee "

    strFrom = "From testing.userid.EMPLOYEE "

    'If the field name is in lowercase, the WHERE
    'clause should be defined as:
    'strWhere = "Where firstname = 'MAUDE' or salary < 25000 "

    strWhere = "Where FIRSTNME='MAUDE'  or SALARY < 25000 "

    strSQL = strSelect & strFrom & strWhere
    If Con.Connectto("Sybase System 10 & 11", strUserid, strPassword,
strSrvName) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
    End If
    Con.Disconnect

      'The second method for connecting to Sybase System 10 Server:

    If Con.Connectto("ODBC Data Sources", strUserid, strPassword, strSrvName &
"!" & strDSname) Then
        Qry.SQL = strSQL
        Set Qry.Connection = Con
        Set RS.Query = Qry
        If RS.Execute Then
            If RS.Numrows <> 0 Then
                Msgbox "Number of row(s) in the resultset = " & RS.NumRows
                RS.Close
            End If
        End If
```

```
    End If
    Con.Disconnect

End Sub
```

```
' Count property
' Access the main table through the data object.

Dim MnTbl As String
Dim NumTbls As Integer
Dim Tbls As Variant
Dim t As Variant

MnTbl = CurrentView.MainTable        ' Get the name of the main table.
Set Tbls = CurrentDocument.Tables    ' Get the collection of tables.
NumTbls = Tbls.Count                 ' Find out how many tables there are.
For i = 1 To NumTbls Step 1
    If (Tbls(i-1).TableName = MnTbl) Then   ' If the right one, then
        ' data access code goes here.
        j=1
    End If
Next
```

```
' CreateDate property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    'If the document has been modified...
    If (CurrentDocument.Modified) Then
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' CurrentPageNum property
' Determine the current page number for the
' current form or form letter.
Dim CrntPage As Integer
CrntPage = CurrentApplication.ActiveView.CurrentPageNum

' Or

' Change the page of the current form or form letter
' to page 2.
CurrentApplication.ActiveView.CurrentPageNum = 2
```

```
' CurrentRecord property
' Find the current record for the active view (form,
' report, worksheet, and so on.)
Dim CrntRec As Long
CrntRec = CurrentApplication.ActiveDocWindow.CurrentRecord

' Or

' Go to a specific record by setting the CurrentRecord property.
' Go to record 100.
CurrentApplication.ActiveDocWindow.CurrentRecord = 100
```

```
'CurrentRow property
'This script program demonstrates the basics of using the Approach
'Data Object (ADO). First, it connects to a SQL Server. If that
'succeeds, it then initializes a Query object.

Dim Conn As New connection     'Create a connection object.
Dim Qry As New query           'Create a Query object.
Dim RsltSet As New ResultSet   'Create a ResultSet object.
Dim Rval, Row, i As Integer
Dim Data As String

'Connect to Microsoft SQL Server
If (Conn.ConnectTo("SQL Server","userid","password","sqlsvr-nt")) Then
'If the connection succeeds, initialize the Query object named Qry.
   Qry.SQL = "SELECT test.authors.au_lname, test.authors.au_fname, test.titles.title,
test.titles.price _
      FROM test.authors, test.titleauthor, test.titles _
      WHERE (test.authors.au_id = test.titleauthor.au_id And test.titleauthor.title_id
= test.titles.title_id) _
      ORDER BY test.authors.au_lname"
'Set the connection property of the query object to use the connection object you
created.
   Set Qry.Connection = Conn
'Set the Query property of the ResultSet object to use the query object you created.
   Set RsltSet.Query = Qry
   If (RsltSet.Execute) Then   'Execute the query.
      NoCols = RsltSet.NumColumns   'If successful, find out how many rows there are.
      Print "====================="
      Do Until (RsltSet.CurrentRow = 10)   'For each row...
         Print "====================="
         Row = Ltrim(RsltSet.CurrentRow)   'Get rid of the spaces.
         Print "Row: " & Row   'Print the row number.
         For i = 1 To NoCols   'For each column.
'Print the field name and then the data.
            Data = Rtrim(Ltrim(RsltSet.FieldName(i))) & ": " &
Ltrim(RsltSet.GetValue(RsltSet.FieldName(i)))
            Print Data
         Next i
         RsltSet.NextRow   'Go to the next row.
         Print "====================="
      Loop
      Print "====================="
   End If
   Conn.Disconnect   'Disconnect from the server.
End If
```

```
' Cyan property
' Find out how much cyan is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Cyan    ' Determine the amount of cyan.
Print b                             ' Print the amount of cyan.
```

```
' DataField property
' Retrieve the name of the table field that the field in the view represents.
Dim Fld As String
Fld = Source.DataField

' Or

' Set the table field that the field in the view represents.
' Note: Make sure the DataTable property contains the correct table.
Source.DataField = "LAST NAME"
```

```
' DataTable property
' Retrieve the name of the table containing the table field that the
' field in the view represents.
Dim Tbl As String
Tbl = Source.DataTable

' Or

' Set the table containing the table field that the field in the
' view represents.
' Customer is a table name.
Source.DataTable = "CUSTOMER"
```

```
' DeleteRow method
'This code is pulled from the 'deleteScheduledRemovedRooms' global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
    Dim Con As New Connection
    Dim Qry As New Query
    Dim ResSet As New ResultSet

    Dim TName As String
    TName = CurrentDocument.Tables(0).TableName

    If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
        Set Qry.Connection = Con
        Qry.TableName = CurrentDocument.Tables(0).Path & TName
            Set ResSet.Query = Qry
            If (ResSet.Execute)Then
                If (ResSet.NumRows) Then
                    ResSet.FirstRow
                    Do
                        ResSet.DeleteRow
                    Loop While (ResSet.NumRows)
                End If
            End If
        Con.Disconnect
    End If
End Sub
```

```
' Description property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    ' If the document has been modified...
    If (CurrentDocument.Modified) Then
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
'Disconnect method
'This code is from the 'deleteScheduledRemovedRooms' global
'function in the Meeting Room Scheduler SmartMaster application.
'It deletes all of the rows in the specified table.

Sub deleteScheduledRemovedRooms
   'Initialize variables for the connection.
   Dim Con As New Connection()
   Dim Qry As New Query()
   Dim ResSet As New ResultSet()
   'Create a shorter way to identify the table.
   Dim TName As Table
   'TName identifies the first (counting from zero) table attached
   'to this .APR file.
   Set TName = CurrentDocument.Tables(0)

   If Con.ConnectTo("dBASE IV") Then   'If the connection is successful, then.
      'Associate this Connection object with the new Query.
      Set Qry.Connection = Con
      Qry.TableName = TName.FullName
      'Associate this Quesry object with the new ResultSet.
      Set ResSet.Query = Qry
         If (ResSet.Execute)Then      'If the ResultSet is successful, then.
            If (ResSet.NumRows) Then   'If the ResultSet isn't empty, then.
               ResSet.FirstRow
               'Delete the rows in the table.
               Do
                  ResSet.DeleteRow
               Loop   While (ResSet.NumRows)
            End If
         End If
      'Close the connection.
      Con.Disconnect      'Shortest correct syntax.
      'Call Con.Disconnect() is more formal syntax.
      'Success = Con.Disconnect() is also correct.
   End If
End Sub
```

```
' Dispatch property
' This script returns the methods and properties for the Microsoft Access
' month OCX object.

Dim myvar as Variant
Set myvar = currentview.Body.objoleobj.Dispatch
myvar.nextmonth
```

```
' Documents property
' Close the first document opened in the active Approach session,
' without knowing the document name.
' The first document is always 0 (zero).

Call CurrentApplication.Documents(0).Window.Close()
```

```
' Document property
' Each view has a document (.APR file) that is its parent.
' Retrieve the document in the current active document window.
' This script will print the name of the .APR file that
' contains the current view.

Print CurrentView.Document.Name
```

```
' DoMenuCommand method
' The script programmer has access to all of the menus in Approach.
' This is useful if you want a script that interacts with the user.
' For instance, you can put the user in Find mode.

CurrentApplication.ApplicationWindow.DoMenuCommand(IDM_FIND)
```

```
' DrillDownView property
' Specify the view that displays detailed data when the user drills down
' from a crosstab.

CurrentDocument.Crosstab~ 1.DrillDownView = CurrentDocument.Chart~ 1
```

```
' Enabled property
' This example comes from the list box lbxRooms in the Room Setup dialog box
' of the Meeting Room Scheduler SmartMaster application.
' If the user selects an item in the list box, the Remove button
' is enabled. If nothing is selected, the Remove button is disabled.

Sub Click(Source As Listbox, X As Long, Y As Long)
     If source.text <> "" Then
          Source.btnRemove.Enabled = True
     Else
          Source.btnRemove.Enabled = False
     End If

End Sub
```

```
' ExcludeFirst property
' Create and run a find for duplicate records.
Sub FindDupes
   'Create a FindDuplicate object.
   Set DupFind = New FindDuplicate("LastName")
   'Leave the first duplicated record out of the found set.
   Dim SuccessFlag As Integer
   SuccessFlag = DupFind.ExcludeFirst

   'Create a Find object.
   Set MyFind = New Find()

   'Attach the FindDuplicate object to the Find object.
   Set MyFind.FindSpecial = DupFind

   'Run the find.
   Call CurrentDocument.Window.FindSort(MyFind)

End Sub
```

```
' Execute method
'This code is from the 'deleteScheduledRemovedRooms' global
'function in the Meeting Room Scheduler SmartMaster application.

Sub deleteScheduledRemovedRooms
     Dim Con As New Connection
     Dim Qry As New Query
     Dim ResSet As New ResultSet

     Dim TName As String
     TName = CurrentDocument.Tables(0).TableName

     If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Qry.Connection = Con
          Qry.TableName = CurrentDocument.Tables(0).Path & TName
               Set ResSet.Query = Qry
               If (ResSet.Execute)Then
                    If (ResSet.NumRows) Then
                         ResSet.FirstRow
                         Do
                              ResSet.DeleteRow
                         Loop While (ResSet.NumRows)
                    End If
               End If
          Con.Disconnect
     End If
End Sub
```

```
' Expand property
' Expand the display element to print all its data.
Source.Expand = True

' Or

' Retrieve the current setting of the Expand property for
' the current display element.
Dim rvalExpnd As Integer
rvalExpnd = Source.Expand
```

```
' FieldNames property
Sub TableReport(TblName As String)
    'This function prints a report to the Output panel on the
    'table whose name is passed in.
    Dim Tbl As Table
    Dim i As Variant

    'Get the table as a Table object.
    Set Tbl=CurrentDocument.GetTableByName(TblName)
    'Since this might be a SQL or Notes table, first
    'find out if it is still connected to the document (.APR file).
    If (Tbl.IsConnection) Then
        'Print the tablename.
        Print "Table name: " & Tbl.TableName
        Print "File name: " & Tbl.FileName
        Print "Full name: " & Tbl.FullName
        Print "Path: " & Tbl.Path
        Print "# of Fields: " & Str(Tbl.NumFields)
        Print "# of Records: " & Str(Tbl.NumRecords)
        Print "====================="
        'The array is zero-based, so we need to start at 0
        'and end at NumFields-1.
        For i = 0 To Tbl.NumFields-1 Step 1
            'The FieldNames array holds the names of the fields.
            Print "Field: " & Tbl.FieldNames(i)
            Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
            Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
            Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
            Print "====================="
        Next
    End If
End Sub
```

```
'FieldName method
'This script prints the name of each column in a result set.
Dim Con As New Connection   'New Connection object
Dim Qry As New Query        'New Query object
Dim RS As New ResultSet     'New ResultSet object
Dim TName As String         'Table to open

'Build the parts of the Query and ResultSet objects.
TName = "orders.dbf"
Qry.TableName = "C:\94orders\" & TName
Set Qry.Connection = Con
Set RS.Query = Qry

'Open the connection.
If Con.ConnectTo("dBASE IV") Then
   'Create the result set.
   If RS.Execute Then
      'Loop through the columns in the result set.
      For i = 1 To RS.NumColumns
            Print RS.FieldName(i)
      Next
   End If
   Con.Disconnect
End If
```

```
' FileName property
Sub DocumentReport()
    ' This script prints a report of document information
    ' to the Output panel.

    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    ' If the document has been modified
    If (CurrentDocument.Modified) Then
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables associated with the .APR: " &
Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' FillField method
' This script fills the EntryDate field in the Customer database
' with Today's date.
    Dim rval As Integer  ' Return value
    rval = CurrentWindow.FillField("Customer.EntryDate", Today)
If (rval) Then
   MessageBox("Filled Field successfully.")
Else
   MessageBox("Fill Field failed.")
End If
```

```
' FindSort method

Sub SampleFinds
   'This sub expects the main table for the current view to have the following fields:
   '   Country      Text
   '   OrderTotal   Numeric

   Call CurrentWindow.FindAll

   'This script shows a correct way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneField As New Find
   Call sfindOneField.And("Country","Japan, USA")
   Call CurrentWindow.FindSort(sfindOneField)

   'This script shows an alternate way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneFieldAlternate As New Find
   Call sfindOneFieldAlternate.And("Country","Japan")
   Call sfindOneFieldAlternate.Or("Country","USA")
   Call CurrentWindow.FindSort(sfindOneFieldAlternate)

   'This script finds multiple values in multiple fields.
   'It finds records with either "Japan" or "USA" in the Country field with Order
Total >= 1000
   'and it also finds records with only "Japan" in the Country field with Order Total
<1000.
   'The two sets of results are returned as the found set.
   Dim sfindComplex As New Find
   Call sfindComplex.And("Country", "Japan, USA")
   Call sfindComplex.And("OrderTotal",">= 1000")
   Call sfindComplex.Or("Country", "Japan")
   Call sfindComplex.And("OrderTotal","<1000")
   Call CurrentWindow.FindSort(sfindComplex)

   'This script shows an INCORRECT way to finding multiple values in the same field.
   'Do not do this by mistake.
   Dim sfindBad As New Find
   Call sfindBad.And("Country","Japan")
   Call sfindBad.And("Country","USA")      'The value "USA" replaces the previous
value.
   Call CurrentWindow.FindSort(sfindBad)
   'The find returns only records with Country = "USA".

   ' This script sorts records according to the values in two different fields.
   'If finds all records, then sorts them in ascending order alphabetically by the
values
   'in the Country field, and then sorts them in descending order numerically by the
```

```
    'values in the OrderTotal field.
    Call CurrentWindow.FindAll
    Dim ssortAll As New sort
    Call ssortAll.Add("Country", LtsSortAscending)
    Call ssortAll.Add("OrderTotal", LtsSortDescending)
    Call CurrentWindow.FindSort(ssortAll)

End Sub
```

```
' FindSpecial property
' Create and run a find for duplicate records.
Sub FindDupes
   'Create a FindDuplicate object.
   Set DupFind = New FindDuplicate("LastName")
   'Leave the first duplicated record out of the found set.
   Dim SuccessFlag As Integer
   SuccessFlag = DupFind.ExcludeFirst

   'Create a Find object.
   Set MyFind = New Find()

   'Attach the FindDuplicate object to the Find object.
   Set MyFind.FindSpecial = DupFind

   'Run the find.
   Call CurrentDocument.Window.FindSort(MyFind)

End Sub
```

```
' FirstRecord method
' Go to the first record.
CurrentApplication.ActiveDocWindow.FirstRecord()
```

```
' FirstRow method
'This code is pulled from the 'deleteScheduledRemovedRooms' global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
    Dim Con As New Connection
    Dim Qry As New Query
    Dim ResSet As New ResultSet

    Dim TName As String
    TName = CurrentDocument.Tables(0).TableName

    If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
        Set Qry.Connection = Con
        Qry.TableName = CurrentDocument.Tables(0).Path & TName
            Set ResSet.Query = Qry
            If (ResSet.Execute)Then
                If (ResSet.NumRows) Then
                    ResSet.FirstRow
                    Do
                        ResSet.DeleteRow
                    Loop While (ResSet.NumRows)
                End If
            End If
        Con.Disconnect
    End If
End Sub
```

```
' FontName property
' Retrieve the font name from the current display element.
Dim Fnt As String
Fnt = Source.Font.FontName

' Or

' Set the font for the current display element.
Source.Font.FontName = "Arial"
```

```
' Font property
' Return the size of the font on the Remove button.
Dim Size As Integer
Size = btnRemove.Font.Size

' Or

' Set the Remove button text to 10 points.
btnRemove.Font.Size = 10
```

```
' FullName property
Sub DocumentReport()
    ' This script prints a report of document information
    ' to the Output panel.

    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    ' If the document has been modified
    If (CurrentDocument.Modified) Then
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables associated with the .APR: " &
Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
'GetAt method
'Retrieve a find or sort condition from a Find, FindDistinct, FindDuplicate,
FindTopLowest, or Sort object.

Sub FindYokoSmith
   'This sub finds a specific name in the Names database with
   'LastName, FirstName, and State fields.

   'Create a find object.
   'Declare the Find object with the first find condition.
   Dim FindA As New Find("Names.LastName","Smith")
   'Add a second find condition in an AND relationship with the first condition.
   Call FindA.And("Names.FirstName","Yoko")
   'Add a third find condition in an AND relationship with the first condition.
   Call FindA.And("Names.State","WA")

   'Print the find conditions.
   Dim Top As Integer       'Top bound of an array
   Dim i As Integer         'Index of an array
   Dim Outfield As String   'Temp name of field
   Dim Condition As String  'Temp condition expression
   Dim AndOr As Integer     'Temp ORNumber

   Top = FindA.GetCount     'Return the number of find conditions.
   Print "The find conditions are"
   Print "Field", "Condition", "Find Level"
   'Loop through the Find object to print each condition.
   For i = 0 To Top-1
      Call FindA.GetAt(i,Outfield,Condition,AndOr)
      Print Outfield, Condition, Str$(AndOr)

      'For a FindDistinct or FindDuplicate object, print each condition as follows:
      'Call FindA.GetAt(i,Outfield)
      'Print Outfield

      'For a FindTopLowest object, print each condition as follows:
      'Call FindA.GetAt(Outfield, NumRecordsSought, HowFound)
      'Print Outfield, NumRecordsSought, Str$(HowFound)

      'For a Sort object, print each condition as follows:
      'Call SortA.GetAt(i, Outfield, SortOrder)
      'Print Outfield, SortOrder

   Next

   'Run the find.
   Call CurrentDocument.Window.FindSort(FindA)
```

```
End Sub
```

```
' GetColorFromRGB method
' Change the background color of the current field based on the value.

If (Val(Source.Text) < 0) Then
' Create a color object using the GetColorFromRGB function.
' Pass in a long value representing a color.  There are Approach constants
' defined for many colors that can be passed to the function.
    Set Source.Background.Color = CurrentApplication.GetColorFromRGB(COLOR_RED)
Elseif (Val(Source.Text) = 0) Then
    Set Source.Background.Color = CurrentApplication.GetColorFromRGB(COLOR_BLUE)
Elseif (Val(Source.Text) > 0) Then
    Set Source.Background.Color = CurrentApplication.GetColorFromRGB(COLOR_GREEN)
Else
    Set Source.Background.Color = CurrentApplication.GetColorFromRGB(COLOR_WHITE)
End If
```

```
'GetCount method
'Retrieve the number of find conditions in a Find object.

Sub FindYokoSmith
   'This sub finds a specific name in the Names database with
   'LastName, FirstName, and State fields.

   'Create a find object.
   'Declare the Find object with the first find condition.
   Dim FindA As New Find("Names.LastName","Smith")
   'Add a second find condition in an AND relationship with the first condition.
   Call FindA.And("Names.FirstName","Yoko")
   'Add a third find condition in an AND relationship with the first condition.
   Call FindA.And("Names.State","WA")

   'Print the find conditions.
   Dim Top As Integer       'Top bound of an array
   Dim i As Integer         'Index of an array
   Dim Outfield As String   'Temp name of field
   Dim Condition As String  'Temp condition expression
   Dim AndOr As Integer     'Temp ORNumber

   Top = FindA.GetCount     'Return the number of find conditions.
   Print "The find conditions are"
   Print "Field", "Condition", "Find Level"
   'Loop through the Find object to print each condition.
   For i = 0 To Top-1
      Call FindA.GetAt(i,Outfield,Condition,AndOr)
      Print Outfield, Condition, Str$(AndOr)
   Next

   'Run the find.
   Call CurrentDocument.Window.FindSort(FindA)
End Sub
```

```
' GetFieldFormula
Sub CalcTableReport
    'This function prints a report to the Output panel on the
    'table of calculated fields in the current document.
    Dim Tbl As Table
    Dim i As Variant

    'Get the calculated field table as a Table object.
    Set Tbl=CurrentDocument.CalcTable
    Print "Calculated fields in " & CurrentDocument.Name
    Print "# of Calculated Fields: " & Str(Tbl.NumFields)
    Print "====================="
    'The array is zero-based, so we need to start at 0
    'and end at NumFields-1.
    For i = 0 To Tbl.NumFields-1 Step 1
        'The FieldNames array holds the names of the fields.
        Print "Field: " & Tbl.FieldNames(i)
        Print "Formula: " & Tbl.GetFieldFormula(Tbl.FieldNames(i))
        Print "====================="
    Next
End Sub
```

```
' GetFieldOptions method
Sub TableReport(TblName As String)
     'This function prints a report to the Output panel on the
     'table whose name is passed in.
     Dim Tbl As Table
     Dim i As Variant

     'Get the Customer table as a Table object.
     Set Tbl=CurrentDocument.GetTableByName("customer")
          'Print the tablename
          Print "Table name: " & Tbl.TableName
          Print "File name: " & Tbl.FileName
          Print "Full name: " & Tbl.FullName
          Print "Path: " & Tbl.Path
          Print "# of Fields: " & Str(Tbl.NumFields)
          Print "# of Records: " & Str(Tbl.NumRecords)
          Print "====================="
          'The array is 0 based, so we need to start at 0
          'and end at NumFields-1.
          For i = 0 To Tbl.NumFields-1 Step 1
               'The FieldNames array holds the names of the fields.
               Print "Field: " & Tbl.FieldNames(i)
               Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
               Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
               Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
               Print "====================="
          Next
End Sub
```

```
' GetFieldSize method
Sub TableReport(TblName As String)
     'This function prints a report to the Output panel on the
     'table whose name is passed in.
     Dim Tbl As Table
     Dim i As Variant

     'Get the Customer table as a Table object.
     Set Tbl=CurrentDocument.GetTableByName("customer")
          'Print the tablename
          Print "Table name: " & Tbl.TableName
          Print "File name: " & Tbl.FileName
          Print "Full name: " & Tbl.FullName
          Print "Path: " & Tbl.Path
          Print "# of Fields: " & Str(Tbl.NumFields)
          Print "# of Records: " & Str(Tbl.NumRecords)
          Print "====================="
          'The array is zero-based, so we need to start at 0
          'and end at NumFields-1.
          For i = 0 To Tbl.NumFields-1 Step 1
               'The FieldNames array holds the names of the fields.
               Print "Field: " & Tbl.FieldNames(i)
               Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
               Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
               Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
               Print "====================="
          Next
End Sub
```

```
' GetFieldType method
Sub TableReport(TblName As String)
    'This function prints a report to the Output panel on the
    'table whose name is passed in.
    Dim Tbl As Table
    Dim i As Variant

    'Get the Customer table as a Table object.
    Set Tbl=CurrentDocument.GetTableByName("customer")
        'Print the tablename
        Print "Table name: " & Tbl.TableName
        Print "File name: " & Tbl.FileName
        Print "Full name: " & Tbl.FullName
        Print "Path: " & Tbl.Path
        Print "# of Fields: " & Str(Tbl.NumFields)
        Print "# of Records: " & Str(Tbl.NumRecords)
        Print "====================="
        'The array is zero-based, so we need to start at 0
        'and end at NumFields-1.
        For i = 0 To Tbl.NumFields-1 Step 1
            'The FieldNames array holds the names of the fields.
            Print "Field: " & Tbl.FieldNames(i)
            Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
            Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
            Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
            Print "====================="
    End If
End Sub
```

```
' GetHandle method
' The GetHandle function retrieves the Windows handle for the
' DocWindow. This comes in handy when you are making API
' calls to Windows that require the WindowHandle.
Dim hWnd As Long
hWnd = CurrentWindow.GetHandle
' You can now call a Window API requiring a handle to a Window
' and pass it to hWnd.
```

```
' GetTableByName method
Sub TableReport(TblName As String)
     ' This script prints a report to the output window on the
     ' table whose name is passed in.
     Dim Tbl As Table
     Dim i As Variant

     ' Get the Customer table as a table object.
     Set Tbl=CurrentDocument.GetTableByName("customer")
          'Print the tablename
          Print "Tablename: " & Tbl.Tablename
          Print "FileName: " & Tbl.Filename
          Print "FullName: " & Tbl.FullName
          Print "Path: " & Tbl.Path
          Print "# of Fields: " & Str(Tbl.NumFields)
          Print "# of Records: " & Str(Tbl.NumRecords)
          Print "====================="
          ' The array is 0 based, so we need to start at 0
          ' and end at NumFields-1.
          For i = 0 To Tbl.NumFields-1 Step 1
               'The FieldNames array holds the names of the fields.
               Print "Field: " & Tbl.FieldNames(i)
               Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
               Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
               Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
               Print "====================="
          Next
End Sub
```

```
' GetText method
' Retrieve the value of the current row in the LastName field
' in the current view, which is a worksheet.
Dim Txt As String
Txt = CurrentApplication.ActiveView.GetText("LastName")
```

```
' GetValue method
' This example creates a result set with all of the fields and records
' from a table and prints the record values.
Dim Con As New Connection
Dim Qry As New Query
Dim RS As New ResultSet
Dim TName As String
Dim i As Integer
TName = CurrentDocument.Tables(0).TableName
Qry.TableName = CurrentDocument.Tables(0).Path & TName
Set Qry.Connection = Con
Set RS.Query = Qry
If Con.ConnectTo("dBASE IV") Then
   If RS.Execute Then
      Do
         For i = 1 To RS.NumColumns
            Print RS.FieldName(i) & " : " & RS.GetValue(i)
         Next
      Loop While RS.NextRow
   End If
End If
```

```
' Green property
' Find out how much green is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Green   ' Determine the amount of green.
Print b                             ' Print the amount of green.
```

```
' GroupByDataField property
' This example creates a report, adds a summary panel to the report that is
' grouped by the field SALESREP in the INVOICE table, and sets the summary
' panel PageBreak property to true so that a new page is created for
' each distinct SALESREP value.

Dim Rpt As Report
Dim Spanel As SummaryPanel
Dim FB As FieldBox
Dim SumFB As FieldBox

Set Rpt = New Report (CurrentDocument, "INVOICE")
Rpt.Name="MyReport"
Set Spanel = New SummaryPanel(CurrentDocument.MyReport.Body)
Spanel.Height=360
Spanel.Background.Color.SetRGB(COLOR_VANILLA)
Spanel.GroupByDataTable="INVOICE"
Spanel.GroupByDataField="SALESREP"

Spanel.PageBreak=True

Spanel.Name="BySalesRep"
Set FB=New Fieldbox (CurrentDocument.MyReport.BySalesRep)
FB.Width=1440
FB.Height=360
FB.DataTable="INVOICE"
FB.DataField="SALESREP"
CurrentDocument.MyReport.Body.Height=0

Set SumFB= New FieldBox(CurrentDocument.MyReport.BySalesRep)
SumFB.Width=1440
SumFB.Height=360
SumFB.Left=2880
SumFB.DataTable="INVOICE"
SumFB.DataField="QtybyRep"
```

```
' GroupByDataTable property
' This example creates a report, adds a summary panel to the report that is
' grouped by the field SALESREP in the INVOICE table, and sets the
' summary panel PageBreak property to true so that a new page is
' created for each distinct SALESREP value.

Dim Rpt As Report
Dim Spanel As SummaryPanel
Dim FB As FieldBox
Dim SumFB As FieldBox

Set Rpt = New Report (CurrentDocument, "INVOICE")
Rpt.Name="MyReport"
Set Spanel = New SummaryPanel(CurrentDocument.MyReport.Body)
Spanel.Height=360
Spanel.Background.Color.SetRGB(COLOR_VANILLA)
Spanel.GroupByDataTable="INVOICE"
Spanel.GroupByDataField="SALESREP"

Spanel.PageBreak=True

Spanel.Name="BySalesRep"
Set FB=New Fieldbox (CurrentDocument.MyReport.BySalesRep)
FB.Width=1440
FB.Height=360
FB.DataTable="INVOICE"
FB.DataField="SALESREP"
CurrentDocument.MyReport.Body.Height=0

Set SumFB= New FieldBox(CurrentDocument.MyReport.BySalesRep)
SumFB.Width=1440
SumFB.Height=360
SumFB.Left=2880
SumFB.DataTable="INVOICE"
SumFB.DataField="QtybyRep"
```

```
' Height property
' Example from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
' Display the reservation owner in the correct time slot
' in the current view body.
' Called from readBlock
   '  txt                  Reservation owner
   '  start                Reservation start time
   '  finish               Reservation end time
   '  roomName             Reserved room's name or number

' * RUNTIME DEPENDENCIES
' * Constants: Use constants defined by LotusScript in
' * LSCONST.LSS.
' * Globals: Use the global array Rooms() filled by the readBlock
' * sub.

' Declare variables
    Dim tt As textbox ' New text block to hold the reservation
                      ' owner's name in the view
    Dim i As Integer  ' Index of array with the room names

' Index of the room that matches the roomName passed in.
' Determine the vertical placement of the reservation in the view.
    Dim matchedRoom As Integer

' Offset and multiplier for the vertical placement of the
' reservation.
    Dim verticalPlacement As Integer

' Search through the global array Rooms to find the room passed
' in from the schedule database using the sub readBlock.
' Set matchedRoom to the index of the room passed in.
    For i = 0 To Ubound(Rooms)
        If Rooms(i) = roomName Then
            matchedRoom = i
            i = Ubound(Rooms)
        End If   ' If element matches the room passed in.
    Next

' Set position and display for the reservation.

' Header in the view takes up 1635 twips. The height of each row in the table
' is 330 twips.
    verticalPlacement = 1635 + (330 * matchedRoom)
```

```
' Create the text block to hold the reservation.
    Set tt = New TextBox(Currentview.Body)

' Fill the text block with the reservation owner's name
' and spaces to center the text properly.
    tt.Text = " " + txt + " "

' Set display properties for the text block to match the form.
    tt.Font.Size = 8
      ' Use LotusScript constants for border style.
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
      ' Use Approach constants for line width.
    tt.Border.Width = $apr1point
      ' Use LotusScript constants for color.
    Call tt.Border.Color.SetRGB(COLOR_ULTRAMARINE)
    Call tt.Background.Color.SetRGB (COLOR_50_GRAY)

' Set the position of the text block to correspond to the
' correct room and time.
    tt.Height = 325
    tt.Top = verticalPlacement   ' Current offset from top of
                                 ' form

' Convert reservation time (passed in) to the horizontal
' location and length on the form.
    tt.Left = (((start - 8) * 750) + 945)
    tt.Width =  (750 * (finish - start))

' Add a prefix to the name of the text block so the clearDisplay
' function can delete the reservation.
    tt.Name = "tt" + Str$(tt.Top) + Str$(tt.Left)

End Sub   ' displayBlock
```

```
' HideMargins property
' Find out if the margins are hidden in the current view.
Dim rval As Integer
rval = CurrentApplication.ActiveView.HideMargins

' Or

' Make the margins for the current view visible.
CurrentApplication.ActiveView.HideMargins = False
```

```
' IconBarVisible property
Sub CleanScreen()
    ' This script performs the same function
    ' as the Clean Screen menu item on the Edit menu.
    CurrentWindow.Redraw=False   ' Turn off Redraw temporarily
    ' Turn off each bar.
    CurrentWindow.ActionBarVisible=False
    CurrentWindow.IconBarVisible=False
    CurrentWindow.StatusBarVisible=False
    CurrentWindow.ViewTabVisible=False
    CurrentWindow.Redraw=True   ' Turn Redraw back on
    CurrentWindow.Repaint       ' Now repaint the window.
End Sub
```

```
' IconSets property
' Find out what icon sets are available.
Dim IcnSets As Variant
Dim i as integer
' Store the icon set collection.
IcnSets = CurrentApplication.IconSets
' Loop through the collection and print the name of each icon set.
ForAll IconSets in IcnSets
   Print IcnSets(i)   ' Icon set name appears in the Output panel of the IDE.
   i = i + 1
End ForAll
```

```
' InsertAfter method
' Insert the ObjButton button behind the LastName display element
' in the layer order.
' After this statement is executed, LastName displays on top of ObjButton.
Source.ObjButton.InsertAfter(Source.LastName)
```

```
' IsChecked property
' Determine if a check box is checked, and then display a message that
' includes the checked or unchecked value.
' This script is placed in an event script for an object in the same
' view as the check box.
If (Source.ObjCheckBox.IsChecked) Then
    MessageBox("The check box is checked and its value is " &
Source.ObjCheckBox.CheckedValue)
Else
    MessageBox("The check box is not checked and its value is " &
Source.ObjCheckBox.UnCheckedValue)
End If
```

```
' IsCommandChecked method
' Determine if the View - Show Tab Order menu item is checked.

Dim rval As Integer
rval = CurrentApplication.ApplicationWindow.IsCommandChecked(IDM_SHOWTABS)
```

```
' IsCommandEnabled method
' Determine if the File - Save menu item is enabled.

Dim rval As Integer
rval = CurrentApplication.ApplicationWindow.IsCommandEnabled(IDM_SAVE)
```

```
' IsEmpty method
' In this example, create a list box and fill
' the list box with the views available.
Dim LstBox As ListBox
Dim aryVws(20) As String
Dim NumVws As Integer

' Check to find out if the current view is a form.
    If (CurrentView.Type = $aprForm) Then
        ' First find out if the collection of views is empty.
        If (CurrentDocument.Views.IsEmpty=False) Then
            NumVws = CurrentDocument.Views.Count   ' Get the number of views.
            For i = 0 To NumVws-1   ' Fill the array with the view names.
                aryVws(i) = CurrentDocument.Views(i).Name
            Next
        End If
    Else
        Messagebox "You must be on a form."
    End If
```

```
' Italic property
' Find out if the font in the current display element is italicized.
' If it is, display a message
If(Source.Font.Italic) Then
   Messagebox("The font is italic.")
Else
   Messagebox("The font is not italic")
End If

' Or

' Set the font of the current display element to italic.
Source.Font.Italic = True
```

```
' Keywords property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' LabelAlignment property
' Change the label so it is aligned with the center of the field.
Source.LabelAlignment = $LtsAlignmentHorizCenter

' Or

' Retrieve the current label alignment setting.
Dim LblAlign As Long
LblAlign = Source.LabelAlignment
```

```
' LabelFont property
' Change the font of the label for the current object to look like the font
' of the LASTNAME object.

Set Source.LabelFont = Source.LASTNAME.LabelFont

' Or

' Create your own font, set the properties, and use it to set the label font.
Dim Fnt as Font
Set Fnt = Source.Labelfont
Fnt.Bold = True             'Bold the font.
Fnt.FontName = "Arial"      'Set the font to Arial.
Fnt.Size = 10               'Set the size to 10 points.
```

```
' LabelPosition property
' Display the label below the object.
Source.LabelPosition = $LtsPositionBottom

' Or

' Retrieve the current position of the label for this object.
Dim LblPos As Long
LblPos = Source.LabelPosition
```

```
' LabelText property
' Set the label for the current field in the view to Last Name.
Source.LabelText = "Last Name"

' Or

' Retrieve the label for the field in the view.
Dim Lbl As String
Lbl = Source.LabelText
```

```
' LastModified property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' LastRecord method
' Go to the last record.
CurrentApplication.ActiveDocWindow.LastRecord()
```

```
' LastRow method
'This code is from the deleteScheduledRemovedRooms global
'function in the Meeting Room Scheduler SmartMaster application.
'It deletes all of the values from a table in a batch process using
'Connection, Query, and ResultSet objects.

Sub deleteScheduledRemovedRooms
   Dim Con As New Connection
   Dim Qry As New Query
   Dim RS As New ResultSet

   Dim TName As String
   TName = CurrentDocument.Tables(0).TableName

   If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
      Set Qry.Connection = Con
      Qry.TableName = CurrentDocument.Tables(0).Path & TName
         Set RS.Query = Qry
         If (RS.Execute) Then
            If (RS.NumRows) Then
               RS.LastRow   'Go to the last row.
            End If
         End If
      Con.Disconnect
   End If
End Sub
```

```
' LeftMargin property
'Find out what the current left margin is (in TWIPS) for the active view.
Dim LeftMrgn As Integer
LeftMrgn = CurrentApplication.ACTIVEVIEW.LeftMargin

'Or

'Set the left margin for the active view to a new value.
CurrentApplication.ACTIVEVIEW.LeftMargin = 1440
```

```
' Left (Border) property
' This script comes from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application.
' Create a new text block with a 1 point left and right ultramarine border.

Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    t = 1635 & (330 * t)
    Set tt = New textbox(currentview.Body)
    tt.text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $ItsBorderThick
    tt.Border.Color.SetRGB (color_ultramarine)
End Sub
```

```
' Left property
' Example from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
' Display the reservation owner in the correct time slot
' in the current view body.
' Called from readBlock
   '  txt                  reservation owner
   '  start                reservation start time
   '  finish               reservation end time
   '  roomName             name or number of reserved room

' * RUNTIME DEPENDENCIES
' * Constants: Uses constants defined by LotusScript defined in
' * LSCONST.LSS.
' * Globals: Uses the global array Rooms() filled by the readBlock
' * sub.

' Declare variables
    Dim tt As textbox    ' New text block to hold the reservation
                                          ' owner's name in the view
    Dim i As Integer     ' Index of array with the room names

' Index of the room that matches the roomName passed in.
' Determine the vertical placement of the reservation in the view.
    Dim matchedRoom As Integer

' Offset and multiplier for the vertical placement of the
' reservation.
    Dim verticalPlacement As Integer

' Search through the global array Rooms to find the room passed
' in from the Schedule database using the sub readBlock.
     ' Set matchedRoom to the index of the room passed in.
    For i = 0 To Ubound(Rooms)
        If Rooms(i) = roomName Then
            matchedRoom = i
            i = Ubound(Rooms)
        End If   ' If element matches the room passed in.
    Next

' Set position and display for the reservation.

' Header in the view takes up 1635 twips, each row in the table
' is 330 twips tall.
    verticalPlacement = 1635 + (330 * matchedRoom)
```

```
' Create the text block to hold the reservation.
    Set tt = New textbox(currentview.body)

' Fill the text block with the reservation owner's name
' and spaces to center the text properly.
    tt.Text = " " + txt + " "

' Set display properties for the text block to match the form.
    tt.Font.Size = 8
      ' Use LotusScript constants for border style.
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
      ' Use Approach constants for line width.
    tt.Border.Width = $apr1point
      ' Use LotusScript constants for color.
    Call tt.Border.Color.SetRGB(COLOR_ULTRAMARINE)
    Call tt.Background.Color.SetRGB (COLOR_50_GRAY)

' Set the position of the text block to correspond to the
' correct room and time.
    tt.Height = 325
    tt.Top = verticalPlacement   ' Current offset from top of
                                                          ' form

' Convert reservation time (passed in) to the horizontal
' location and length on the form.
    tt.Left = (((start - 8) * 750) + 945)
    tt.Width =  (750 * (finish - start))

' Add a prefix to the name of the text block so the clearDisplay
' function can delete the reservation.
    tt.Name = "tt" + Str$(tt.Top) + Str$(tt.Left)

End Sub   ' displayBlock
```

```
' LineStyle property
' Reference the Pattern property of the line using the LineStyle property.
Sub Click(Source As Button, X As Long, Y As Long, Flags As Long)
    Source.ObjLine.LineStyle.Pattern = $ltsLineStyleDot
End Sub
```

```
'ListDataSources method
'Prints the names of the data source types available from this machine.
Sub PrintDataSourceNames
   Dim Con As New connection      'A new connection object
   Dim ListOfData As Variant      'The resulting list
   Dim i As Integer               'Loop index

   'Create the array of data source types.
   ListOfData = Con.ListDataSources()
   'Print each item in the array to Output.
   For i = 0 To Ubound(ListOfData)-1
      Print ListOfData(i)
   Next
End Sub
```

```
'ListFields method
'Prints the names of the fields in a table of a data source type
'corresponding to a Connection object.
Sub PrintFieldList
   Dim Con As New Connection   'A new Connection object
   Dim FieldList As Variant    'Resulting list of fields
   Dim i As Integer            'Index for print loop

   'Connect to a data source.
   Call Con.ConnectTo("dBase IV")

   'Read fields from a table of the same data source.
   FieldList = Con.ListFields("orders.dbf")

   'Print each item in the array to Output.
   For i = 1 To Ubound(FieldList)
      Print FieldList(i)
   Next
End Sub
```

```
'ListTables method
'First of two examples:
'Prints the names of the tables available from this server.
Sub PrintTableNamesFromODBC
   Dim Con As New Connection      'A new connection object
   Dim ListOfTables As Variant    'The resulting list
   Dim i As Integer               'Loop index

   'Create a connection to the database Sample.
   Call Con.ConnectTo("ODBC Data Sources", "UserID", "Password", "!Sample")
   'Create the array of table names.
   ListOfTables = Con.ListTables()
   'Print each item in the array to Output.
   For i = 0 To Ubound(ListOfTables)-1
      Print ListOfTables(i)
   Next
   Con.Disconnect
End Sub

'Second of two examples:
'Prints the tables available from this directory.
Sub PrintTableNamesFromDirectory
   Dim Con As New Connection      'A new connection object
   Dim ListOfTables As Variant    'The resulting list
   Dim i As Integer               'Loop index

   'Create a connection to Paradox databases.
   Call Con.ConnectTo("Paradox")
   'Create the array of table names.
   ListOfTables = Con.ListTables("c:\lotus\work\approach\")
   'Print each item in the array to Output.
   For i = 0 To Ubound(ListOfTables)-1
      Print ListOfTables(i)
   Next
   Con.Disconnect
End Sub
```

```
' MacroClick property
' Set the Main Menu button to run a macro when it is clicked.
MainMenuButton.MacroClick = "Go To Main Menu"
```

```
' MacroDataChange property
' Run the Go To Customer Information macro when the data in the
' current field changes.
Source.MacroDataChange = "Go To Customer Information"

' Or

' Get the name of the macro that is set to run if the data changes in the
' current field.
Dim mstr As String
mstr = Source.MacroDataChange
```

```
' MacroTabIn property
' Run the macro Go To Customer Information when
' the user tabs into the current display element.
Source.MacroTabIn = "Go To Customer Information"

' Or

' Get the name of the macro that is currently set to
' run when the user tabs into the current display element.
Dim MyMacroName As String
MyMacroName = Source.MacroTabIn

' Or

' Assign the macro to another display element.
Source.Field2.MacroTabIn = Source.MacroTabIn
```

```
' MacroTabOut property
' Run the macro Go To Customer Information when the
' user tabs out of the current display element.
Source.MacroTabOut = "Go To Customer Information"

' Or

' Get the name of the macro that is currently set to
' run when the user tabs out of the current display element.
Dim MyMacroName As String
MyMacroName = Source.MacroTabOut

' Or

' Assign the macro to another display element.
Source.Field2.MacroTabOut = Source.MacroTabOut
```

```
' Magenta property
' Find out how much magenta is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Magenta ' Determine the amount of magenta.
Print b                             ' Print the amount of magenta.
```

```
' MainTable property
' Access the main table through the data object.
' The first step is to find out where the main table is.
Dim MnTbl As String
Dim NumTbls As Integer
Dim Tbls As Variant
Dim t As Variant

MnTbl = CurrentView.MainTable      ' Get the name of the main table.
Set Tbls = CurrentDocument.Tables  ' Get the collection of tables.
NumTbls = Tbls.Count               ' Find out how many tables there are.
For i = 1 To NumTbls Step 1
    If (Tbls(i-1).TableName = MnTbl) Then   ' If we've found the right one.
        j=1
    End If
Next
```

```
' MakeNamedStyle method
' Create a named style called MyStyle based on the attributes
' of the current object.
Dim MyReturnValue As Integer
MyReturnValue = Source.MakeNamedStyle("MyStyle")
```

```
' Maximize method
' After opening a document, you can expand the window
' to use the full size of the screen, so no other applications are visible.

CurrentApplication.ApplicationWindow.Maximize()
```

```
' MenuBar property
' Change the menu for the current view to
' a custom menu named EasyMenu.

Dim Mnu As Variant
Dim NumMnus As Integer

Mnu = CurrentApplication.Menus      ' Get the list of menus.

NumMnus = Ubound(Mnu)               ' Find out how many custom menus there are.
' Arrays are 0 based.
For i = 0 To NumMnus Step 1
     If (Mnu(i) = "EasyMenu") Then  ' If the right menu is available.
          CurrentView.MenuBar = Mnu(i)    ' Change it.
          i = NumMnus
     Elseif (i = NumMnus) Then      ' If the right menu isn't there.
          Messagebox "EasyMenu is not available.",,"Menus"
     End If
Next
```

```
' Menus
' Print the names of the menus available.
Sub PrintMenuNames
   Dim MenuNames As Variant      ' The resulting list of menus
   Dim i As Integer              ' Loop index

   'Create the array of menus.
   MenuNames = Document.Menus
   'Print each item in the array to Output.
   For i = 0 To Ubound(MenuNames)
      Print MenuNames(i)
   Next
End Sub
```

```
' Minimize method
' After opening a document, you can hide Approach so it runs in the
' background.

CurrentApplication.ApplicationWindow.Minimize()
```

```
' Modified property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    'If the document has been modified...
    If (CurrentDocument.Modified) Then
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' NamedFindSorts property
' Change the named find/sort to the "CA" named find/sort
Dim Finds As Variant
Dim NumFinds As Integer

Finds = CurrentDocument.NamedFindSorts

NumFinds = Ubound(Finds)       ' Find out how many named finds there are.
' Arrays are 0 based.
For i = 0 To NumFinds Step 1
     If (Finds(i) = "CA") Then   ' If we've found the right named find/sort.
          CurrentWindow.NamedFindSort = Finds(i)   'Change it.
     Elseif (i = NumFinds) Then   ' If the right named find/sort isn't there.
          Messagebox "The CA named find/sort is not available.",,"Find/Sort"
     End If
Next
```

```
' NamedFindSort property
' Change the named find/sort to the "CA" named find/sort
Dim Finds As Variant
Dim NumFinds As Integer

Finds = CurrentDocument.NamedFindSorts

NumFinds = Ubound(Finds)       ' Find out how many named finds there are.
' Arrays are 0 based.
For i = 0 To NumFinds Step 1
    If (Finds(i) = "CA") Then   ' If we've found the right named find/sort.
        CurrentWindow.NamedFindSort = Finds(i)   ' Change it.
    Elseif (i = NumFinds) Then   ' If the right named find/sort isn't there.
        Messagebox "The CA named find/sort is not available.",,"Find/Sort"
    End If
Next
```

```
' NamedStyles property
' Change the named style for the current display element to
' the CompanyStyle named style.

Dim Styl As Variant
Dim NumStyls As Integer

Styl = CurrentDocument.NamedStyles   ' Get the list of named styles.

NumStyls = Ubound(Styl)    ' Find out how many named styles there are.
'Arrays are 0 based.
For i = 0 To NumStyls Step 1
     If (Styl(i) = "CompanyStyle") Then ' If we've found the right named style.
          Source.NamedStyle = Styl(i)    ' Change it.
          i = NumStyls
     Elseif (i = NumStyls) Then         ' If the right named style isn't there.
          Messagebox "The company named style is not available.",,"Styles"
     End If
Next
```

```
' NamedStyle property
' Change the named style for the current display element or panel
' to the CompanyStyle named style.

Dim Styl As Variant
Dim NumStyls As Integer

Styl = CurrentDocument.NamedStyles   'Get the list of named styles.

NumStyls = Ubound(Styl)    'Find out how many named styles there are.
' Arrays are 0 based.
For i = 0 To NumStyls Step 1
   If (Styl(i) = "CompanyStyle") Then   'If you find the right one.
      Source.NamedStyle = Styl(i)       'Change it.
      i = NumStyls
   Elseif (i = NumStyls) Then           'If the named style isn't there.
      Messagebox "The company named style is not available.",,"Styles"
   End If
Next
```

```
' Name property
' Example from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
' Display the reservation owner in the correct time slot
' in the current view body.
' Called from readBlock
   '  txt                  reservation owner
   '  start                reservation start time
   '  finish               reservation end time
   '  roomName             name or number of reserved room
' * RUNTIME DEPENDENCIES
' * Constants: Uses constants defined by LotusScript defined in
' * LSCONST.LSS.
' * Globals: Uses the global array Rooms() filled by the readBlock
' * sub.

' Declare variables
    Dim tt As textbox    ' New text block to hold the reservation
                                          ' owner's name in the view
    Dim i As Integer     ' Index of array with the room names

' Index of the room that matches the roomName passed in.
' Determine the vertical placement of the reservation in the view.
    Dim matchedRoom As Integer

' Offset and multiplier for the vertical placement of the
' reservation.
    Dim verticalPlacement As Integer

' Search through the global array Rooms to find the room passed
' in from the Schedule database using the sub readBlock.
     ' Set matchedRoom to the index of the room passed in.
    For i = 0 To Ubound(Rooms)
        If Rooms(i) = roomName Then
            matchedRoom = i
            i = Ubound(Rooms)
        End If   ' If element matches the room passed in.
    Next

' Set position and display for the reservation.

' Header in the view takes up 1635 twips, each row in the table
' is 330 twips tall.
    verticalPlacement = 1635 + (330 * matchedRoom)

' Create the text block to hold the reservation.
```

```
    Set tt = New textbox(currentview.body)

' Fill the text block with the reservation owner's name
' and spaces to center the text properly.
    tt.Text = " " + txt + " "

' Set display properties for the text block to match the form.
    tt.Font.Size = 8
      ' Use LotusScript constants for border style.
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
      ' Use Approach constants for line width.
    tt.Border.Width = $apr1point
      ' Use LotusScript constants for color.
    Call tt.Border.Color.SetRGB(COLOR_ULTRAMARINE)
    Call tt.Background.Color.SetRGB (COLOR_50_GRAY)

' Set the position of the text block to correspond to the
' correct room and time.
    tt.Height = 325
    tt.Top = verticalPlacement   ' Current offset from top of
                                                          ' form

' Convert reservation time (passed in) to the horizontal
' location and length on the form.
    tt.Left = (((start - 8) * 750) + 945)
    tt.Width =  (750 * (finish - start))

' Add a prefix to the name of the text block so the clearDisplay
' function can delete the reservation.
    tt.Name = "tt" + Str$(tt.Top) + Str$(tt.Left)

End Sub   ' displayBlock
```

```
' NewPage method
' Add a new page to the current form.
CurrentApplication.ActiveView.NewPage
```

```
' NewRecord method
' Create a new record.

CurrentApplication.ActiveDocWindow.NewRecord()
```

```
' New (Button) method
' Create a new button.
    Dim Btn As Button
    ' Create a new Click Here button in the current view.
    Set Btn = New Button(CurrentApplication.ActiveView.Body,"Click Here")
    Btn.Left = 2000   'Position the button, in twips.
    Btn.Top = 2000
    Btn.Width = 1750
    Btn.Height = 640
    Btn.NonPrinting = True   'Do not print this button.
    ' Set the attributes using a pre-existing named style.
    Btn.NamedStyle = "Default"
```

```
' New (ChartView) method
' Create a new chart.
' In the example below, the Salesrep field is grouped on the x-axis;
' the Qty field is summed on the y-axis; and a legend label is created
' for each distinct value in the Category field.  The x, y, and s
' arguments are arrays of strings.
' The enumeration constants - $aprCalcSum and $aprChartTypeBar - signify
' that the values represented by the y argument should be summed and that
' the chart is a bar chart. Precede each enumeration constant with a $.
' After you create the chart the script makes the new chart the active view
' and names it.
' The context for this example might be a button in an existing view.

Dim x(0) As String
Dim y(0) As String
Dim s(0) As String

x(0)= "Salesrep"
y(0) = "Qty"
s(0)= "Category"

Dim chrt As Chartview
Set chrt = New ChartView(currentDocument, x, y, s, $aprCalcSum, $aprChartTypeBar)

Set currentWindow.activeView=chrt
chrt.Name="MyCHRT"
```

```
' New (Checkbox) method
' This script creates a new check box.
' It assumes an existing table named Customer with a field
' named Type and an existing named style of MyStyle.
Dim ChkBox as CheckBox   'Declare the CheckBox object.
Set ChkBox = New CheckBox(CurrentView.Body)  'Create a new check box on the current
view on page 1 (default page).
' Define the settings for the check box.
ChkBox.Left = 400   ' In twips
ChkBox.Top = 400    ' In twips
' Set the attributes using a pre-existing named style.
ChkBox.NamedStyle = MyStyle
' Set the label for the check box.
ChkBox.LabelText = "Distributor"
' Assign the check box to the Customer table.
ChkBox.DataTable = "Customer"
' Assign the check box to the Type field.
ChkBox.DataField = "Type"
' Set the default value of the check box as checked.
ChkBox.SetState = True
```

```
' New (Collection) method
' Collections let you make a group of objects.
Dim rval As Integer   ' Return value
Dim CustomerColl As Collection

Set CustomerColl = New Collection()

rval = CustomerColl.Add(Source.FirstName)
rval = CustomerColl.Add(Source.LastName)
rval = CustomerColl.Add(Source.Address)
rval = CustomerColl.Add(Source.City)
rval = CustomerColl.Add(Source.State)
rval = CustomerColl.Add(Source.Postal_Cod)
rval = CustomerColl.Add(Source.Company)

' Copy the Company field to the third object
' of the collection.
rval = CustomerColl.SetAt(3, Source.Company)

' The only fields needed are FirstName,
' LastName, and Company. Remove the remaining
' fields from the collection. Since the SetAt
' function copies the object to the specified location,
' two copies of the Company field exist in the collection.
' Start at the end of the collection and work
' backwards to get rid of the Company field at the
' end of the collection.
' Arrays are zero-based.
i = CustomerColl.Count - 1
Do While i >2
     If (CustomerColl(i).Name = "COMPANY") Then
          CustomerColl.Remove(i+1)
          ' Because the object was deleted, reduce
          ' the number of items in the collection by 1.
          i = i - 1
     End If
     If (CustomerColl(i).Name = "ADDRESS") Then
          CustomerColl.Remove(i+1)
          i = i - 1
     End If
     If (CustomerColl(i).Name = "CITY") Then
          CustomerColl.Remove(i+1)
          i = i - 1
     End If
     If (CustomerColl(i).Name = "STATE") Then
          CustomerColl.Remove(i+1)
          i = i - 1
```

```
    End If
    If (CustomerColl(i).Name = "POSTAL_COD") Then
        CustomerColl.Remove(i+1)
        i = i - 1
    End If
Loop
```

```
' New (Color) method
' Create a new color

Dim MyRed As Color
Set MyRed = New Color(224, 31, 36)

' Set the background color to MyRed.
Source.Background.Color = MyRed
```

```
' New (Crosstab) method
' The example below creates a new crosstab using the fields
' Salesrep, Category, and Qty for the rows, columns, and body of the crosstab,
' respectively. The arguments represented by R, C, and B are arrays of
' strings.

' Enumeration constants ($aprCalcSum) designate the summary
' calculation type for the body cells, as well as for the row and column
' totals. Each enumeration constant is preceded by $ (dollar sign). String
' values (Total) label the grand summary row and column of the
' crosstab.
' The arguments for row, column, and body are arrays of strings. Thus,
' you can create multilevel summaries, and you can also add
' more than one field to the body.

Dim R(1) As String
Dim C(1) As String
Dim B(1) As String

R(1) = "Salesrep"
C(1) = "Category"
B(1) = "Qty"

Dim CR As Crosstab

Set CR = New Crosstab(CurrentDocument,R,C,B,$aprCalcSum,"Total",$aprCalcSum,"Total",
$aprCalcSum)
```

```
' New (Document) method
' Creates a new document.
Dim Con As New Connection
Dim Qry As New Query
Dim ResSet As New ResultSet
Dim Doc As Document

If Con.ConnectTo("dBASE IV") Then   ' Connect to dBASE.
     Set Qry.Connection = Con
     Qry.TableName = "C:\LOTUS\APPROACH\CUSTOMER.DBF"
     Set ResSet.Query = Qry
     If (ResSet.Execute)Then
          Set Doc = New Document(ResSet)
     End If
     Con.Disconnect
End If
```

```
' New (DropDownBox) method
' Create a drop-down box in the current view, on page 1.
Sub Click(Source As Button, X As Long, Y As Long, Flags As Long)
    Dim DropDnBox As DropDownBox
    Set DropDnBox = New DropDownBox(Source.Parent,1)
End Sub
```

```
' New (FieldBox) method
' Create a new field box on the current form, on page 1. After you create
' the field box, bind it to a field in the database by specifying
' the DataTable and DataField properties.
Sub Click(Source As Button, X As Long, Y As Long, Flags As Long)
    Dim FldBox As FieldBox

    Set FldBox = New FieldBox(Source.Parent,1)
    FldBox.DataTable = "Customer"
    FldBox.DataField = "Company"
End Sub
```

```
' New method (FindDistinct)
'Create and run a find for distinct records without other conditions.
Sub DistinctRecords
   'Create a FindDistinct object.
   Set DisFind = New FindDistinct("Country")

   'Run the find.
   Call CurrentWindow.FindSort(DisFind)
End Sub

' Create and run a find for distinct records as part of a Find object.
Sub PostalCodeFind
   'Create a FindDistinct object.
   Set DisFind = New FindDistinct("PostalCode")

   'Create a Find object.
   Set MyFind = New Find()

   'Add a condition to find all records in Japan.
   Call MyFind.And("Country", "Japan")

   'Attach the FindDistinct object to the Find object.
   Set MyFind.FindSpecial = DisFind

   'Run the find.
   Call CurrentWindow.FindSort(MyFind)

End Sub
```

```
' New method (FindDuplicate)
' Create and run a find for distinct records without other conditions.
Sub DuplicateRecords
   'Create a FindDuplicate object.
   Set DupFind = New FindDuplicate("LastName")

   'Run the find.
   Call CurrentWindow.FindSort(DupFind)
End Sub

' Create a FindDuplicate object as part of a Find object.
Sub FindDups
   'Create a FindDuplicate object.
   Set DupFind = New FindDuplicate("LastName")

   'Create a Find object.
   Set MyFind = New Find()

   'Add a condition to find all records in Japan.
   Call MyFind.And("Country", "Japan")

   'Attach the FindDistinct object to the Find object.
   Set MyFind.FindSpecial = DupFind

   'Run the find.
   Call CurrentWindow.FindSort(MyFind)

End Sub
```

```
' New method (Find)
Sub SampleFinds
   'This sub expects the main table for the current view to have the following fields:
   '   Country      Text
   '   OrderTotal   Numeric

   Call CurrentWindow.FindAll

   'This script shows a correct way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneField As New Find
   Call sfindOneField.And("Country","Japan, USA")
   Call CurrentWindow.FindSort(sfindOneField)

   'This script shows an alternate way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneFieldAlternate As New Find
   Call sfindOneFieldAlternate.And("Country","Japan")
   Call sfindOneFieldAlternate.Or("Country","USA")
   Call CurrentWindow.FindSort(sfindOneFieldAlternate)

   'This script finds multiple values in multiple fields.
   'It finds records with either "Japan" or "USA" in the Country field with Order
Total >= 1000
   'and it also finds records with only "Japan" in the Country field with Order Total
<1000.
   'The two sets of results are returned as the found set.
   Dim sfindComplex As New Find
   Call sfindComplex.And("Country", "Japan, USA")
   Call sfindComplex.And("OrderTotal",">= 1000")
   Call sfindComplex.Or("Country", "Japan")
   Call sfindComplex.And("OrderTotal","<1000")
   Call CurrentWindow.FindSort(sfindComplex)

   'This script shows an INCORRECT way to finding multiple values in the same field.
   'Do not do this by mistake.
   Dim sfindBad As New Find
   Call sfindBad.And("Country","Japan")
   Call sfindBad.And("Country","USA")      'The value "USA" replaces the previous
value.
   Call CurrentWindow.FindSort(sfindBad)
   'The find returns only records with Country = "USA".

   ' This script sorts records according to the values in two different fields.
   'If finds all records, then sorts them in ascending order alphabetically by the
values
   'in the Country field, and then sorts them in descending order numerically by the
   'values in the OrderTotal field.
```

```
    Call CurrentWindow.FindAll
    Dim ssortAll As New sort
    Call ssortAll.Add("Country", LtsSortAscending)
    Call ssortAll.Add("OrderTotal", LtsSortDescending)
    Call CurrentWindow.FindSort(ssortAll)

End Sub
```

```
' New (Form) method
' This script creates a new form and adds a field box to the new form.
' The field box is then set to display the values from a field in the
' current table.

Dim Frm As Form
Dim FB As Fieldbox

Set Frm = New Form(CurrentDocument)
Set CurrentWindow.ActiveView=Frm
Frm.Name="MyForm"

Set FB=New Fieldbox (currentdocument.myform.body)
FB.width=1440
FB.height=360
FB.datatable="INVOICE"
FB.datafield="SALESREP"
```

```
' New (LineObject) method
' Create a new line on the current form on page #1.

Sub Click(Source As Button, X As Long, Y As Long, Flags As Long)
    Dim L As LineObject

    Set L = New LineObject(Source.Parent,1)
    L.Left = 1440
    L.Top = 1440
End Sub
```

```
' New (ListBox)
' This script creates a list box and fills
' it with the names of available views.

Dim LstBox As ListBox
Dim aryVws(20) As String
Dim NumVws As Integer

' Find out if the current view is a form.
If (CurrentView.Type = $aprForm) Then
     ' Create a new list box in the current view.
     Set LstBox = New ListBox(CurrentView.Body, 1)
     NumVws = CurrentDocument.Views.Count ' Get the number of views.
     For i = 0 To NumVws-1                ' Fill the array with the view names.
          aryVws(i) = CurrentDocument.Views(i).Name
     Next
Else
     Messagebox "You must be on a form."
End If
```

```
' New (Picture) method
' This global function is taken from the Viewer
' SmartMaster application.

Function createPictures(f As form) As Integer
    On Error Resume Next

    Dim obj As Integer
    Dim c As collection
    Dim p As picture
    Dim pnl As panel

    Set pnl = f.body
    Set c = f.objectlist
    Forall o In c
        If (o.type = $aprPicture) Then
            Delete o
        End If
    End Forall

    obj = 0
    For i = 1 To ((Ubound(pictureList)/numPicColumns) + 1)
        If i = 1 Then
            t = t + 900 + 270 + 90
        Else
            t = t + 1440 + 270
        End If
        l = 0
        For j = 1 To numPicColumns
            If (j <= (Ubound(pictureList) + 1)/i) Then
                If j = 1 Then
                    l = l + 720 + 270 + 90
                Else
                    l = l + 1440 + 270
                End If
                Set p = New picture(pnl, pictureList(obj))
                p.height = 1440
                p.width = 1440
                p.left = l
                p.top = t
                obj = obj + 1
            End If
        Next
    Next
    runapproachmacro("switchToViewer")
End Function
```

```
' New (RadioButton) method
' Create a new RadioButton object on the current form on page #1. This script
' is part of the Click event of a display element such as a field box.

Sub Click(Source As Button, X As Long, Y As Long, Flags As Long)
    Dim rb As RadioButton
    Set rb = New RadioButton(Source.Parent,1)
End Sub
```

```
' New (Rectangle) method
' Create a new rectangle in the current view on page #1.
   Dim Rect As Rectangle
   Set Rect = New Rectangle(Source.Parent,1)
```

```
' New (Report) method
' Create a new report and designate INVOICE as the main table for that
' report. After you create the new report, make the new report the
' active view and name it. Then add a summary panel that groups records by
' SALESREP to the report and name it.
' Add the fields SALESREP and QtybyRep to the summary panel.
' Set the body height to zero so that only one row appears for each
' SALESREP with a subtotal of QTY for that SALESREP.
' Calculate the QTY subtotal by adding the previously
' defined summary calculation (SSUM of QTY) to the summary panel.

Dim Rpt As Report
Dim Spanel As SummaryPanel
Dim FB As FieldBox
Dim SumFB As FieldBox

Set Rpt = New Report (CurrentDocument, "INVOICE")
Rpt.name="MyReport"
Set Spanel = New SummaryPanel(CurrentDocument.MyReport.body)
Spanel.Height=360
Spanel.background.color.setrgb(COLOR_VANILLA)
Spanel.GROUPBYDATATABLE="INVOICE"
Spanel.GROUPBYDATAFIELD="SALESREP"
Spanel.name="BySalesRep"
Set FB=New Fieldbox (CurrentDocument.MyReport.BySalesRep)
FB.width=1440
FB.height=360
FB.datatable="INVOICE"
FB.datafield="SALESREP"
Set CurrentApplication.ActiveView = CurrentDocument.MyReport
CurrentDocument.MyReport.Body.Height=0

Set SumFB= New FieldBox(CurrentDocument.MyReport.BySalesRep)
SumFB.width=1440
SumFB.height=360
SumFB.LEFT=2880
SumFB.datatable="INVOICE"
SumFB.datafield="QtybyRep"
```

```
' New (RoundRect) method
' Create a new rounded rectangle in the current view on page #1.
   Dim RoundRectangle As RoundRect
   Set RoundRectangle = New RoundRect(Source.Parent,1)
```

```
' New method (Sort)
Sub SampleFinds
   'This sub expects the main table for the current view to have the following fields:
   '   Country      Text
   '   OrderTotal   Numeric

   ' This script sorts records according to the values in two different fields.
   'If finds all records, then sorts them in ascending order alphabetically by the
values
   'in the Country field, and then sorts them in descending order numerically by the
   'values in the OrderTotal field.
   Call CurrentWindow.FindAll
   Dim ssortAll As New Sort
   Call ssortAll.Add("Country", LtsSortAscending)
   Call ssortAll.Add("OrderTotal", LtsSortDescending)
   Call CurrentWindow.FindSort(ssortAll)

End Sub
```

```
' New (SummaryPanel) method
' Create a new report and designate INVOICE as the main table for that
' report. After the new report is created, make the new report
' the active view and name it. Then add a summary panel that groups records by
' SALESREP to the report and name the summary panel.
' After the summary panel is added to the report, add the fields SALESREP and
' QtybyRep to the summary panel. Set the report body height to
' zero so that only one row appears for each SALESREP with a subtotal of QTY
' for that SALESREP. Calculate the QTY subtotal by adding the previously
' defined summary calculaltion (SSUM of QTY) to the summary panel.

Dim Rpt As Report
Dim Spanel As SummaryPanel
Dim FB As FieldBox
Dim SumFB As FieldBox

Set Rpt = New Report (CurrentDocument, "INVOICE")
Rpt.Name="MyReport"
Set Spanel = New SummaryPanel(CurrentDocument.MyReport.Body)
Spanel.Height=360
Spanel.Background.Color.SetRGB(COLOR_VANILLA)
Spanel.GroupByDataTable="INVOICE"
Spanel.GroupByDataField="SALESREP"

Spanel.PageBreak=True

Spanel.Name="BySalesRep"
Set FB=New Fieldbox (CurrentDocument.MyReport.BySalesRep)
FB.Width=1440
FB.Height=360
FB.DataTable="INVOICE"
FB.DataField="SALESREP"
CurrentDocument.MyReport.Body.Height=0

Set SumFB= New FieldBox(CurrentDocument.MyReport.BySalesRep)
SumFB.Width=1440
SumFB.Height=360
SumFB.Left=2880
SumFB.DataTable="INVOICE"
SumFB.DataField="QtybyRep"
```

```
' New (TextBox) method
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    Dim h As Integer
    Dim i As Integer

    For i = 0 To Ubound(rooms)
        If rooms(i) = roomName Then
            t = i
            i = Ubound(rooms)
        End If
    Next
    t = 1635 + (330 * t)
    Set tt = New textbox(currentview.body)
    tt.text = " " & txt & " "
    tt.font.size = 8
    tt.border.style = $ltsBorderStyleNone
    tt.border.left = True
    tt.border.right = True
    tt.border.top = False
    tt.border.bottom = False
    tt.border.width = $apr1point
    tt.border.color.setrgb color_ultramarine
    tt.background.color.setrgb color_50_gray
    tt.height = 325
    tt.top = t
    tt.left = (((start - 8) * 750) + 960)
    tt.width =  (750 * (finish - start))
End Sub
```

```
' New (Worksheet) method
' Create a new worksheet and use the AddColumn method to add the
' field QTY to the new worksheet. Arguments to name the column and
' position the column are optional.
' Context for this example is a click event of a button
' on a form in the current document.

Dim Wrk As Worksheet
Set Wrk = New Worksheet(currendocument)
Set CurrentWindow.ActiveView=Wrk
Wrk.Name="Wrksheet"
Wrk.AddColumn("QTY")
```

```
' NextRecord method
' Go to the next record.

CurrentApplication.ActiveDocWindow.NextRecord()
```

```
' NextRow method
' This example makes a result set from a table associated with the current
' document and prints all of the values in the result set.

Dim RS As New ResultSet
Dim i As Integer
'Create the result set.
Set RS = Currentdocument.Tables(0).CreateResultSet
'Loop through all of the records and print each field name and corresponding value.
Do
   For i = 1 To RS.NumColumns
      Print RS.FieldName(i) & " : " & RS.GetValue(i)
   Next
Loop While RS.NextRow
```

```
' NonPrinting property
' Set the current display element not to print.
Source.NonPrinting = True

' Or

' Get the value of the property.
Dim MyReturnValue As Integer
MyReturnValue = Source.NonPrinting
```

```
' NumColumns method
Dim c As New connection
Dim qu As New query
Dim rs As New resultset
tname = currentdocument.tables(0).tablename
qu.Tablename = currentdocument.tables(0).path & tname
Set qu.connection = c
Set rs.query = qu
If c.connectto("dBASE IV") Then
   If rs.execute Then
      Do
         For i = 1 To RS.NUMCOLUMNS
            Print RS.FIELDNAME(i) & " : " & RS.GETVALUE(i)
         Next
      Loop While RS.NEXTROW
   End If
End If
```

```
' NumFields property
Sub TableReport(TblName As String)
    'This function prints a report to the Output panel on the
    'table whose name is passed in.
    Dim Tbl As Table
    Dim i As Variant

    'Get the table as a Table object.
    Set Tbl=CurrentDocument.GetTableByName(TblName)
    'Since this might be a SQL or Notes table, first
    'find out if it is still connected to the document (.APR file).
    If (Tbl.IsConnection) Then
        'Print the tablename
        Print "Table name: " & Tbl.TableName
        Print "File name: " & Tbl.FileName
        Print "Full name: " & Tbl.FullName
        Print "Path: " & Tbl.Path
        Print "# of Fields: " & Str(Tbl.NumFields)
        Print "# of Records: " & Str(Tbl.NumRecords)
        Print "====================="
        'The array is zero-based, so start at 0
        'and end at NumFields-1.
        For i = 0 To Tbl.NumFields-1 Step 1
            'The FieldNames array holds the names of the fields.
            Print "Field: " & Tbl.FieldNames(i)
            Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
            Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
            Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
            Print "====================="
        Next
    End If
End Sub
```

```
' NumJoins property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' NumPages property
' Find the number of pages in the current view.

Dim NumPgs As Integer
NumPgs = CurrentApplication.ActiveView.NumPages
```

```
' NumRecordsFound property
' Implement a named find/sort, and then print the number of records found.

CurrentWindow.NamedFindSort="CA"
' Find out how many records were found.
Print CurrentWindow.NumRecordsFound
```

```
' NumRecords property
Sub TableReport(TblName As String)
    'This function prints a report to the Output panel on the
    'table whose name is passed in.
    Dim Tbl As Table
    Dim i As Variant

    'Get the table as a table object.
    Set Tbl=CurrentDocument.GetTableByName(TblName)
    'Since this might be a SQL or Notes table, first
    'find out if it is still connected to the document (.APR file).
    If (Tbl.IsConnection) Then
        'Print the tablename
        Print "Table name: " & Tbl.TableName
        Print "File name: " & Tbl.FileName
        Print "Full name: " & Tbl.FullName
        Print "Path: " & Tbl.Path
        Print "# of Fields: " & Str(Tbl.NumFields)
        Print "# of Records: " & Str(Tbl.NumRecords)
        Print "====================="
        'The array is zero-based, so start at 0
        'and end at NumFields-1.
        For i = 0 To Tbl.NumFields-1 Step 1
            'The FieldNames array holds the names of the fields.
            Print "Field: " & Tbl.FieldNames(i)
            Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
            Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
            Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
            Print "====================="
        Next
    End If
End Sub
```

```
' NumRevisions property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' NumRows method
'This code is pulled from the deleteScheduledRemovedRooms global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
     Dim Con As New Connection
     Dim Qry As New Query
     Dim ResSet As New ResultSet

     Dim TName As String
     TName = CurrentDocument.Tables(0).TableName

     If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Qry.Connection = Con
          Qry.TableName = CurrentDocument.Tables(0).Path & TName
               Set ResSet.Query = Qry
               If (ResSet.Execute)Then
                    If (ResSet.NumRows) Then
                         ResSet.FirstRow
                         Do
                              ResSet.DeleteRow
                         Loop While (ResSet.NumRows)
                    End If
               End If
          Con.Disconnect
     End If
End Sub
```

```
' NumTables property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' NumViews property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' OnSwitchFromMacro property
' Retrieve the name of the macro to be triggered when users switch from
' the view.
Dim FrmMacro As String
FrmMacro = CurrentApplication.ActiveView.OnSwitchFromMacro

' Or

' Set the macro to be executed when users switch from the view.
CurrentApplication.ActiveView.OnSwitchFromMacro = "MyMacro"
```

```
' OnSwitchToMacro property
' Retrieve the name of the macro to be triggered when users switch
' to the view.
Dim ToMacro As String
ToMacro = CurrentApplication.ActiveView.OnSwitchToMacro

' Or

' Set the macro to be executed when users switch to the view.
CurrentApplication.ActiveView.OnSwitchToMacro = "MyMacro"
```

```
' OpenDocument method
' This script lays the groundwork for an application that runs
' in the background. After opening a document, hide Approach.

Dim rval As Integer   ' Return Value

' Minimize Approach (optional)
CurrentApplication.ApplicationWindow.Minimize

' Hide Approach from the user
CurrentApplication.Visible=False

' Open an application that runs automatically.
rval = CurrentApplication.OpenDocument("AutoApp", "C:\LOTUS\APPROACH")

' Run your program.

' Show Approach to the user
CurrentApplication.Visible=True

' Maximize Approach (optional)
CurrentApplication.ApplicationWindow.Maximize

' Quit Approach
CurrentApplication.CloseWindow
```

```
' Orientation property
' Change the orientation of the line.  The Orientation property
' accepts Approach constants.
Source.ObjLine.Orientation = $ltsOrientationPosSlope
```

```
' Or method
Sub SampleFinds
   'This sub expects the main table for the current view to have the following fields:
   '   Country      Text
   '   OrderTotal   Numeric

   Call CurrentWindow.FindAll

   'This script shows a correct way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneField As New Find
   Call sfindOneField.And("Country","Japan, USA")
   Call CurrentWindow.FindSort(sfindOneField)

   'This script shows an alternate way to find multiple values in the same field.
   'It finds either "Japan" or "USA" in the Country field.
   Dim sfindOneFieldAlternate As New Find
   Call sfindOneFieldAlternate.And("Country","Japan")
   Call sfindOneFieldAlternate.Or("Country","USA")
   Call CurrentWindow.FindSort(sfindOneFieldAlternate)

   'This script finds multiple values in multiple fields.
   'It finds records with either "Japan" or "USA" in the Country field with Order
Total >= 1000
   'and it also finds records with only "Japan" in the Country field with Order Total
<1000.
   'The two sets of results are returned as the found set.
   Dim sfindComplex As New Find
   Call sfindComplex.And("Country", "Japan, USA")
   Call sfindComplex.And("OrderTotal",">= 1000")
   Call sfindComplex.Or("Country", "Japan")
   Call sfindComplex.And("OrderTotal","<1000")
   Call CurrentWindow.FindSort(sfindComplex)

   'This script shows an INCORRECT way to finding multiple values in the same field.
   'Do not do this by mistake.
   Dim sfindBad As New Find
   Call sfindBad.And("Country","Japan")
   Call sfindBad.And("Country","USA")      'The value "USA" replaces the previous
value.
   Call CurrentWindow.FindSort(sfindBad)
   'The find returns only records with Country = "USA".

   ' This script sorts records according to the values in two different fields.
   'If finds all records, then sorts them in ascending order alphabetically by the
values
   'in the Country field, and then sorts them in descending order numerically by the
   'values in the OrderTotal field.
```

```
    Call CurrentWindow.FindAll
    Dim ssortAll As New Sort
    Call ssortAll.Add("Country", LtsSortAscending)
    Call ssortAll.Add("OrderTotal", LtsSortDescending)
    Call CurrentWindow.FindSort(ssortAll)

End Sub
```

```
' PageBreak property
' This example creates a report, adds a summary panel to the report that is
' grouped by the field SALESREP in the INVOICE table, and sets the
' summary panel PageBreak property to true so that a new page is
' created for each distinct SALESREP value.

Dim Rpt As Report
Dim Spanel As SummaryPanel
Dim FB As FieldBox
Dim SumFB As FieldBox

Set Rpt = New Report (CurrentDocument, "INVOICE")
Rpt.Name="MyReport"
Set Spanel = New SummaryPanel(CurrentDocument.MyReport.Body)
Spanel.Height=360
Spanel.Background.Color.SetRGB(COLOR_VANILLA)
Spanel.GroupByDataTable="INVOICE"
Spanel.GroupByDataField="SALESREP"

Spanel.PageBreak=True

Spanel.Name="BySalesRep"
Set FB=New Fieldbox (CurrentDocument.MyReport.BySalesRep)
FB.Width=1440
FB.Height=360
FB.DataTable="INVOICE"
FB.DataField="SALESREP"
CurrentDocument.MyReport.Body.Height=0

Set SumFB= New FieldBox(CurrentDocument.MyReport.BySalesRep)
SumFB.Width=1440
SumFB.Height=360
SumFB.Left=2880
SumFB.DataTable="INVOICE"
SumFB.DataField="QtybyRep"
```

```
' Page property
' Return the number of the page containing the current display element.
Dim MyPageNumber As Integer
MyPageNumber = Source.Page
```

```
' Parent (Document Class) property
' Determine the path of the Approach executable running Orders.APR

Dim TempAppPath as String
TempAppPath = Orders.Parent.Path
```

```
' Parent property
' Print the name of the parent of this object to the Output panel of
' the IDE.
Print Source.Parent.Parent.Name
' If the current object is a button, the output is
' the name of the view containing the panel that the button is on.
' If the current object is a summary panel, the output is
' the name of the report that the panel is on.
```

```
'Password property
'Sets the password for a server connection (write-only)
Dim C As New Connection
C.UserID = "UserID"
C.Password = "UserPassword"
CkConnectTo = C.ConnectTo("SQL Server", C.UserID, C.Password, "sqlsvr_nt351")
C.Disconnect
```

```
' Path property
Sub DocumentReport()
    ' This script prints a report of document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables associated with the .APR: " &
Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' Pattern property
' Change the style of a line to solid. The Pattern property accepts
' Approach enumerators for the line styles.

Source.ObjLine.LineStyle.Pattern = $ltsLineStyleSolid
```

```
' PrevRecord method
' Go to the previous record.

CurrentWindow.PrevRecord()
```

```
' PrevRow Record
'This code is pulled from the 'deleteScheduledRemovedRooms' global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
     Dim C As New Connection
     Dim Q As New Query
     Dim RS As New ResultSet

     Dim tname As String
     tname = currentdocument.tables(0).tablename

     If C.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Q.Connection = C
          Q.Tablename = currentdocument.tables(0).path & tname
               Set RS.Query = Q
               If (RS.Execute)Then
                    If (RS.numrows) Then
                         RS.lastrow
                         Do
                              RS.deleteRow
                    RS.PrevRow
                         Loop While (RS.numrows)
                    End If
               End If
          c.disconnect
     End If
End Sub
```

```
' Query property
'This code is pulled from the 'deleteScheduledRemovedRooms' global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
     Dim C As New Connection
     Dim Q As New Query
     Dim RS As New ResultSet

     Dim tname As String
     tname = currentdocument.tables(0).tablename

     If C.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Q.Connection = C
          Q.Tablename = currentdocument.tables(0).path & tname
               Set RS.Query = Q
               If (RS.Execute)Then
                    If (RS.numrows) Then
                         RS.firstrow
                         Do
                              RS.deleteRow
                         Loop While (RS.numrows)
                    End If
               End If
          c.disconnect
     End If
End Sub
```

```
' Quit property
' Close the current Approach executable.

CurrentApplication.Quit
```

```
' RadioButtonLabel property
'Retrieve the label for the radio button named ObjRadio.
Dim RBLabel As String
RBLabel  = Source.ObjRadio.LabelText

'Or

'Set the label for the radio button named ObjRadio.
Source.ObjRadio.LabelText = "My Radiobutton Label"
```

```
' ReadOnly property
' Find out if the field named Address is read-only.
If (Source.Address.ReadOnly) Then
   Messagebox("The field is read-only")
Else
   Messagebox("The field is read-write")
Endif

' Or

' Set the Address display element to be read-only.
Source.Address.ReadOnly = True
```

```
' Redraw property
' This property lets you turn redraw off while you move and add objects
' to the form, then redraw the entire form when you are done.

Sub TidyScreen()
    CurrentWindow.Redraw=False          ' Turn off redraw temporarily
    CurrentWindow.LastName.Left = 1440  ' Move the LastName field.
    CurrentWindow.FirstName.Left = 2880 ' Move the FirstName field.
    CurrentWindow.Address.Left = 1440   ' Move the Address field.
    CurrentWindow.City.Left = 1440      ' Move the City field.
    CurrentWindow.Redraw=True           ' Turn redraw back on.
    Call CurrentWindow.Repaint()        ' Now repaint the window.
End Sub
```

```
' Reduce property
' Reduce the display element so it prints only as large as the
' data it contains.
Source.Reduce = True

' Or

' Retrieve the current setting of the Reduce property for
' the current display element.
Dim ExpndProp As Integer
ExpndProp = Source.Reduce
```

```
' Red property
' Find out how much red is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Red     ' Determine the amount of red.
Print b                             ' Print the amount of red.
```

```
' Refresh method
' Resfreshes the document window.

CurrentApplication.ActiveDocWindow.Refresh()
```

```
' Relief property
' Change the check box so it has a raised effect.
Source.ObjCheckBox.Relief = $ltsReliefRaised
```

```
' Remove method
' Collections let you make a group of objects.
Dim rval As Integer   ' Return value
Dim CustomerColl As Collection

Set CustomerColl = New Collection()

rval = CustomerColl.Add(Source.FirstName)
rval = CustomerColl.Add(Source.LastName)
rval = CustomerColl.Add(Source.Address)
rval = CustomerColl.Add(Source.City)
rval = CustomerColl.Add(Source.State)
rval = CustomerColl.Add(Source.Postal_Cod)
rval = CustomerColl.Add(Source.Company)

' Copy the Company field to the third object
' of the collection.
rval = CustomerColl.SetAt(3, Source.Company)

' The only fields needed are FirstName,
' LastName, and Company. Remove the remaining
' fields from the collection. Since the SetAt
' function copies the object to the specified location,
' two copies of the Company field exist in the collection.
' Start at the end of the collection and work
' backwards to get rid of the Company field at the
' end of the collection.
' Arrays are zero-based.
i = CustomerColl.Count - 1
Do While i >2
     If (CustomerColl(i).Name = "COMPANY") Then
          CustomerColl.Remove(i+1)
          ' Because the object was deleted, reduce
          ' the number of items in the collection by 1.
          i = i - 1
     End If
     If (CustomerColl(i).Name = "ADDRESS") Then
          CustomerColl.Remove(i+1)
          i = i - 1
     End If
     If (CustomerColl(i).Name = "CITY") Then
          CustomerColl.Remove(i+1)
          i = i - 1
     End If
     If (CustomerColl(i).Name = "STATE") Then
          CustomerColl.Remove(i+1)
          i = i - 1
```

```
    End If
    If (CustomerColl(i).Name = "POSTAL_COD") Then
        CustomerColl.Remove(i+1)
        i = i - 1
    End If
Loop
```

```
' Repaint method
' This script performs the same function
' as the Clean Screen menu item on the Edit menu.

Sub CleanScreen()
    CurrentWindow.Redraw=False     ' Turn off redraw temporarily
                                   ' Turn off each bar.
    CurrentWindow.ActionBarVisible=False
    CurrentWindow.IconBarVisible=False
    CurrentWindow.StatusBarVisible=False
    CurrentWindow.ViewTabVisible=False
    CurrentWindow.Redraw=True      ' Turn it back on
    Call CurrentWindow.Repaint()        ' Now repaint the window.
End Sub
```

```
' ReplaceWithResultSet method
' This script creates a result set through a Connection object
' and uses the result set to replace a Table object associated with the current
document.

Sub UpdateData
   ' Declare variables to create a result set.
   Dim Con As New Connection
   Dim Qry As New Query
   Dim RS As New Resultset
   Dim RVal As Integer         'Return value flag
   'Declare an array to store a mapping between fields of the table and result set.
   Dim MyPairs(1 To 3,1 To 2) As String

   ' Build the map array.
   MyPairs(1,1)="TableField1"
   MyPairs(1,2)="RSField1"
   MyPairs(2,1)="TableField2"
   MyPairs(2,2)="RSField2"
   MyPairs(3,1)="TableField3"
   ' The result set has only two fields.
   MyPairs(3,2)=""

   If Con.ConnectTo("dBASE IV") Then   'If the connection is successful, then.
      'Associate this Connection object with the new Query.
      Set Qry.Connection = Con
      Qry.SQL="Select * From " & """c:\lotus\work\approach\NewData""" & "NewData"
      'Associate this Query object with the new ResultSet.
      Set Rs.Query = Qry
      If (Rs.Execute)Then       'If the ResultSet is successful, then.
         If (Rs.NumRows) Then   'If the ResultSet isn't empty, then.
            'Replace the existing table with the new result set.
            RVal = CurrentDocument.tables(0).ReplaceWithResultSet(RS, MyPairs)
            Print RVal
         End If
      End If
      'Close the connection.
      Con.Disconnect
   End If
End Sub
```

```
' Restore method
' Restores the document window.
CurrentApplication.ActiveDocWindow.Restore()

' Or

' Restore the Approach Window
CurrentApplication.ApplicationWindow.Restore()
```

```
' RightMargin property
'Find out what the current right margin setting is for the active view.
Dim RMargin As Long
RMargin = CurrentApplication.ActiveView.RightMargin

'Or

'Set the right margin for the active view.
CurrentApplication.ActiveView.RightMargin = 720
```

```
' Right (Border) property
' This script comes from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application.
' Create a new text block with a 1 point left and right ultramarine border.

Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    t = 1635 & (330 * t)
    Set tt = New textbox(currentview.Body)
    tt.text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $ItsBorderThick
    tt.Border.Color.SetRGB (color_ultramarine)
End Sub
```

```
' RunApproachMacro method
'This code example comes from the click event of the CheckRadio object
'in the Data Entry Screen view in the Checkbook SmartMaster application.

Sub Click(Source As Radiobutton, X As Long, Y As Long)
     source.checknumber.visible=True
     source.depositnumber.visible=False
     RunApproachMacro("SetCheckNumber")
End Sub
```

```
' RunProcedure method
' This script is written for a 1-2-3 application that calls Approach to run
' the global function TestPrc. This script will not run in Approach.
' The script assumes that there is an .APR file named TestFile.APR and
' a global sub in that document named TestPrc.

Sub OLETest
   ' Create an instance of Approach.
   Set appApproach = CreateObject("Approach.Application")

   ' Use appApproach as the source for the next statements.
   With appApproach  ' "With" is a shorthand method in scripting

      ' Open the .APR file.
      Call .OpenDocument("TestFile.apr", "c:\lotus\work\approach")

      .visible = true      ' Set the Document to appear on screen; the OLE automation
defaults to hidden.

      ' Execute the global function, passing a string as an argument.
      Call .RunProcedure("TestPrc", "Hello World")
   End With
End Sub
```

```
' SendToBack method
' Place the current object behind all other display elements
' that are in the same area of the form.
Source.SendToBack
```

```
' SetAt method
' Collections let you make a group of objects.
Dim rval As Integer   ' Return value
Dim CustomerColl As Collection

Set CustomerColl = New Collection()

rval = CustomerColl.Add(Source.FirstName)
rval = CustomerColl.Add(Source.LastName)
rval = CustomerColl.Add(Source.Address)
rval = CustomerColl.Add(Source.City)
rval = CustomerColl.Add(Source.State)
rval = CustomerColl.Add(Source.Postal_Cod)
rval = CustomerColl.Add(Source.Company)

' Copy the Company field to the third object
' of the collection.
rval = CustomerColl.SetAt(3, Source.Company)

' The only fields needed are FirstName,
' LastName, and Company. Remove the remaining
' fields from the collection. Since the SetAt
' function copies the object to the specified location,
' two copies of the Company field exist in the collection.
' Start at the end of the collection and work
' backwards to get rid of the Company field at the
' end of the collection.
' Arrays are zero-based.
i = CustomerColl.Count - 1
Do While i >2
     If (CustomerColl(i).Name = "COMPANY") Then
          CustomerColl.Remove(i+1)
          ' Because the object was deleted, reduce
          ' the number of items in the collection by 1.
          i = i - 1
     End If
     If (CustomerColl(i).Name = "ADDRESS") Then
          CustomerColl.Remove(i+1)
          i = i - 1
     End If
     If (CustomerColl(i).Name = "CITY") Then
          CustomerColl.Remove(i+1)
          i = i - 1
     End If
     If (CustomerColl(i).Name = "STATE") Then
          CustomerColl.Remove(i+1)
          i = i - 1
```

```
    End If
    If (CustomerColl(i).Name = "POSTAL_COD") Then
        CustomerColl.Remove(i+1)
        i = i - 1
    End If
Loop
```

```
' SetCellFocus method
' Select the current cell in the current worksheet in the column
' you specify by name: Company. You might attach the following
' script to the CellGetFocus event of a worksheet. It selects
' the current cell in the Company column and then prints the text of that
' cell.

Dim Cell As Integer
Cell=CurrentView.SetCellFocus("Company")
Print Current.View.GetText
```

```
' SetFocus method
' Example from the Enter Date dialog box in the Meeting Room
' Scheduler SmartMaster application.
' Clear the field of text and give it the focus
Sub Switchto(Source As Form, View As VIEW)
     source.body.fbxDate.text = ""
     source.body.fbxDate.setfocus
End Sub
```

```
' SetList method
' This global function, modifyRoomsArray, comes from the Meeting Room
' Scheduler SmartMaster sample application.

Sub modifyRoomsArray(modifyType As Integer)
     Dim fbx As FieldBox
     Dim lbx As ListBox
     Dim btn As Button

     Dim i As Integer
     Dim ret As Integer
     Dim roomExists As Integer
     Dim newRoomName As String
     Dim tempRooms() As String
     Dim numRooms As Integer

     If ((Ubound(rooms) + 1) = 20) And (modifyType > 0)Then
          Messagebox "Cannot add anymore rooms.  Maximum number of rooms is 20."
     Else
          Set fbx = Currentview.Body.fbxRoomName
          Set lbx = Currentview.Body.lbxRooms
          Set btn = Currentview.Body.btnDone
          If (modifyType > 0) Then
               roomExists = False
               newRoomName = fbx.Text
               For i = 0 To Ubound(rooms)
                    If (Ucase$(rooms(i)) = Ucase$(newRoomName)) Then
                         roomExists = True
                         i = Ubound(rooms) + 1
                    End If
               Next
               If (roomExists = False) Then
                    If (Ubound(rooms) = 0  And rooms(0) = "") Then
                         rooms(0) = newRoomName
                    Else
                         Redim Preserve rooms(Ubound(rooms) + 1)
                         rooms(Ubound(rooms)) = newRoomName
                    End If
                    lbx.Setlist rooms
                    fbx.Text = ""
                    btn.Enabled = True
               Else
                    Messagebox "A room named """ & Ucase$(newRoomName) & """ already
exists"
               End If
          Else
```

```
            ret = Messagebox( "Are you sure you want to delete this room?  Deleting
this room will also delete any scheduled conferences for this room.", 4, "Delete
Room")
            If ret = 6 Then
                deleteRoomName = lbx.Text
                Redim Preserve deletedRooms(Ubound(deletedRooms) + 1)
                deletedRooms(Ubound(deletedRooms)) = deleteRoomName
                For i = 0 To Ubound(rooms)
                    If (Ucase$(rooms(i)) = Ucase$(deleteRoomName)) Then
                        numRooms = numRooms - 1
                        For j = i To (Ubound(rooms) - 1)
                            rooms(j) = rooms(j + 1)
                        Next
                        i = Ubound(rooms) + 1
                    End If
                Next
                If (Ubound(rooms)) Then
                    Redim Preserve rooms(Ubound(rooms) - 1)
                Else
                    rooms(0) = ""
                    CurrentView.Body.btnRemove.Enabled = False
                End If
            End If
        End If
        lbx.Setlist rooms
        btn.Enabled = True
        string_sort rooms
    End If

End Sub
```

```
' SetPicture method
' Show the image TILES.BMP in the Win95 directory
' in the Picture object.

Source.ObjPicture.SetPicture("c:\win95\tiles.bmp")
```

```
' SetRGB
' This script comes from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application.

Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As Textbox

    Dim t As Integer
    Dim i As Integer

    For i = 0 To Ubound(rooms)
        If rooms(i) = roomName Then
            t = i
            i = Ubound(rooms)
        End If
    Next
    t = 1635 + (330 * t)
    Set tt = New Textbox(currentview.body)
    tt.Text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $ItsBorderThick
    tt.Border.Color.SetRGB (color_ultramarine)
    tt.Background.Color.SetRGB (color_50_gray)
    tt.Height = 325
    tt.Top = t
    tt.Left = (((start - 8) * 750) + 960)
    tt.Width =  (750 * (finish - start))
    tt.Name = "tt" & Str$(tt.top) & Str$(tt.left)
End Sub
```

```
' SetState method
' This script creates a new check box and sets the default value as checked.
' It assumes an existing table named Customer with a field
' named Type and an existing named style of MyStyle.
Dim ChkBox as CheckBox   'Declare the Checkbox object.
Set ChkBox = New CheckBox(CurrentView.Body)  'Create a new check box on the current
view on page 1 (default page).
' Define the settings for the check box.
ChkBox.Left = 400   ' In Twips
ChkBox.Top = 400    ' In Twips
' Set the attributes using a pre-existing named style.
ChkBox.NamedStyle = MyStyle
' Set the label for the check box.
ChkBox.LabelText = "Distributor"
' Assign the check box to the Customer table.
ChkBox.DataTable = "Customer"
' Assign the check box to the Type field.
ChkBox.DataField = "Type"
' Set the default value of the check box as checked.
Call ChkBox.SetState(True)
```

```
' SetValue
'This is the modifyRooms global function from the Meeting Room Scheduler
'SmartMaster application.

Function modifyRooms
     Dim Con As New Connection
     Dim Qry As New Query
     Dim RS As New ResultSet

     modifyRooms = False

     If Con.ConnectTo("dBASE IV") Then
          Set Qry.Connection = Con
          Qry.Tablename = CurrentDocument.Tables(0).Path & "rooms"
          Set RS.Query = Qry
          If (RS.Execute) Then
               If (RS.NumRows) Then
                    RS.FirstRow
                    Do
                         RS.DeleteRow
                    Loop While (RS.NumRows)
               End If
               modifyRooms = True
               'Loop through the global array "rooms".
               For i = 0 To Ubound(rooms)
                    RS.AddRow
                    RS.SetValue "room", rooms(i)
                    RS.UpdateRow
               Next
          End If
     End If
     Con.Disconnect
End Function
```

```
' ShadowColor property
' Create a color object named Red.
' Set the shadow color of the current display element to the color red.
Dim Red As New color(255,0,0)
Set Source.ShadowColor = Red
```

```
' ShowArrow property
' Displays an arrow on the dropdown box.
Sub Click(Source As Dropdownbox, X As Long, Y As Long)
    Source.CustName.ShowArrow = True
End Sub
```

```
' ShowAsDialog property
' Find out if the form is displayed as a dialog box.
Dim rval As Integer
rval = CurrentApplication.ActiveView.ShowAsDialog

' Or

' Specify that the current form be displayed as a dialog box.
CurrentApplication.ActiveView.ShowAsDialog = True
```

```
' ShowInPreview property
' Set this display element not to show in Print Preview.
Source.ShowInPreview = False

' Or

' Find out if this display element is set to show in Print Preview.
Dim MyReturnValue As Integer
MyReturnValue = Source.ShowInPreview
```

```
' Size property
' Create a new text box and change the font to 8 points.
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As Textbox

    Dim t As Integer
    Dim i As Integer

    For i = 0 To Ubound(rooms)
        If rooms(i) = roomName Then
            t = i
            i = Ubound(rooms)
        End If
    Next
    t = 1635 + (330 * t)
    Set tt = New Textbox(currentview.body)
    tt.text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $apr1point
    tt.Border.Color.SetRGB(color_ultramarine)
    tt.Background.Color.SetRGB(color_50_gray)
    tt.Height = 325
    tt.Top = t
    tt.Left = (((start - 8) * 750) + 960)
    tt.Width =  (750 * (finish - start))
    tt.Name = "tt" & Str$(tt.top) & Str$(tt.left)
End Sub
```

```
' SlideLeft property
' Set this display element to slide to the left if there are extra
' spaces when printing.
Source.SlideLeft = True

' Or

' Find out if this display element is set to slide left when printing.
Dim MyReturnValue As Integer
MyreturnValue = Source.SlideLeft
```

```
' SlideUp property
' Set this display element to slide up if there are extra
' spaces when printing.
Source.SlideUp = True

' Or

' Find out if this display element is set to slide up when printing.
Dim MyReturnValue As Integer
MyReturnValue = Source.SlideUp
```

```
' SQL property
'This code is pulled from the deleteScheduledRemovedRooms global
'function in the Schedule application.

Sub deleteScheduledRemovedRooms
     Dim Con As New Connection
     Dim Qry As New Query
     Dim ResSet As New ResultSet

     Dim TName As String
     TName = CurrentDocument.Tables(0).TableName

     If Con.ConnectTo("dBASE IV") Then   'Connect to dBASE.
          Set Qry.Connection = Con
          Qry.TableName = CurrentDocument.Tables(0).Path & TName
          For i = 1 To Ubound(DeletedRooms)
               Qry.SQL = "SELECT * FROM """ & Qry.TableName & """" & TName & " WHERE ("&
TName & "."Room Name/Number"" = '" & DeletedRooms(i) & "')"
               Set ResSet.Query = Qry
               If (ResSet.Execute)Then
                    If (ResSet.NumRows) Then
                         ResSet.FirstRow
                         Do
                              ResSet.DeleteRow
                         Loop While (ResSet.NumRows)
                    End If
               End If
          Next
          Con.Disconnect
     End If
End Sub
```

```
' StatusBarVisible property
Sub CleanScreen()
    ' This script performs the same function
    ' as the Clean Screen menu item on the Edit menu.
    CurrentWindow.Redraw=False   ' Turn off redraw temporarily
    ' Turn off each bar.
    CurrentWindow.ActionBarVisible=False
    CurrentWindow.IconBarVisible=False
    CurrentWindow.StatusBarVisible=False
    CurrentWindow.ViewTabVisible=False
    CurrentWindow.Redraw=True   ' Turn redraw back on
    CurrentWindow.Repaint       ' Now repaint the window.
End Sub
```

```
' Strikethrough property
' Find out if the current display element has strikethrough text.
Dim rval As Integer
rval = Source.Font.StrikeThrough

' Or

' Set the current display element to have strikethrough text.
Source.Font.StrikeThrough = True
```

```
' Style property
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    Dim t As Integer
    Dim i As Integer

    For i = 0 To Ubound(rooms)
        If rooms(i) = roomName Then
            t = i
            i = Ubound(rooms)
        End If
    Next
    t = 1635 + (330 * t)
    Set tt = New textbox(currentview.body)
    tt.text = " " & txt & " "
    tt.font.size = 8
    tt.border.style = $ltsBorderStyleNone
    tt.border.left = True
    tt.border.right = True
    tt.border.top = False
    tt.border.bottom = False
    tt.border.width = $ItsBorderThick
    tt.border.color.setrgb color_ultramarine
    tt.background.color.setrgb color_50_gray
    tt.height = 325
    tt.top = t
    tt.left = (((start - 8) * 750) + 960)
    tt.width =  (750 * (finish - start))
    tt.name = "tt" & Str$(tt.top) & Str$(tt.left)
End Sub
```

```
'TableName method (Query class)
'This script prints the name of each column in a result set.
Dim Con As New Connection   'New Connection object
Dim Qry As New Query        'New Query object
Dim RS As New ResultSet     'New ResultSet object
Dim TName As String         'Table to open

'Build the parts of the Query and ResultSet objects.
TName = "orders.dbf"
Qry.TableName = "C:\94orders\" & TName
Set Qry.Connection = Con
Set RS.Query = Qry

'Open the connection.
If Con.ConnectTo("dBASE IV") Then
   'Create the result set.
   If RS.Execute Then
      'Loop through the columns in the result set.
      For i = 1 To RS.NumColumns
            Print RS.FieldName(i)
      Next
   End If
   Con.Disconnect
End If
```

```
' TableName property (Table class)
Sub TableReport(TblName As String)
    'This function prints a report to the Output panel on the
    'table whose name is passed in.
    Dim Tbl As Table
    Dim i As Variant

    'Get the table as a Table object.
    Set Tbl=CurrentDocument.GetTableByName(TblName)
    'Since this might be a SQL or Notes table, first
    'find out if it is still connected to the document (.APR file).
    If (Tbl.IsConnection) Then
        'Print the tablename.
        Print "Table name: " & Tbl.TableName
        Print "File name: " & Tbl.FileName
        Print "Full name: " & Tbl.FullName
        Print "Path: " & Tbl.Path
        Print "# of Fields: " & Str(Tbl.NumFields)
        Print "# of Records: " & Str(Tbl.NumRecords)
        Print "====================="
        'The array is zero-based, so start at 0
        'and end at NumFields-1.
        For i = 0 To Tbl.NumFields-1 Step 1
            'The FieldNames array holds the names of the fields.
            Print "Field: " & Tbl.FieldNames(i)
            Print "Options: " & Tbl.GetFieldOptions(Tbl.FieldNames(i))
            Print "Size: " & Str(Tbl.GetFieldSize(Tbl.FieldNames(i)))
            Print "Type: " & Str(Tbl.GetFieldType(Tbl.FieldNames(i)))
            Print "====================="
        Next
    End If
End Sub
```

```
' Tables property
' We need to access the main table through the data object.
' The first step is to find out the name of the main table.
Dim MnTbl As String
Dim NumTbls As Integer
Dim Tbls As Variant
Dim t As Variant

MnTbl = CurrentView.MainTable       ' Get the name of the main table.
Set Tbls = CurrentDocument.Tables   ' Get the collection of tables.
NumTbls = Tbls.Count                ' Find out how many tables there are.
For i = 1 To NumTbls Step 1
    If (Tbls(i-1).TableName = MnTbl) Then   ' If we've found the right table.
        ' Data access code goes here
        j=1
    End If
Next
```

```
' TabNext
' Sets focus to the next field in the Tab Order.
CurrentWindow.TabNext()
```

```
' TabOrder property
' Set the tab order for this display element.
Source.TabOrder = 5

' Or

' Print the value of the tab order of this display element to the output window
' of the IDE.
Print Source.TabOrder
```

```
' TabPrev method
' Sets focus to the previous field in the Tab Order.
CurrentWindow.TabPrev()
```

```
' TabStop property
' Set this display element to allow tabbing.
Source.TabStop = True

' Or

' Return the value of this property.
Dim MyReturnValue As Integer
MyReturnValue = Source.TabStop
```

```
' Text (Button) property
' Find out what text displays on the button ObjButton.
Dim Txt As String
Txt = Source.ObjButton.Text

' Or

' Set the text to display on the button ObjButton.
Source.ObjButton.Text = "Push Me"
```

```
' Text property
' Create a new text box showing the name of each room.
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    Dim t As Integer
    Dim i As Integer

    For i = 0 To Ubound(rooms)
        If rooms(i) = roomName Then
            t = i
            i = Ubound(rooms)
        End If
    Next
    t = 1635 + (330 * t)
    Set tt = New textBox(CurrentView.Body)
    tt.text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $apr1point
    tt.Border.Color.SetRGB (color_ultramarine)
    tt.Background.Color.SetRGB (color_50_gray)
    tt.Height = 325
    tt.Top = t
    tt.Left = (((start - 8) * 750) + 960)
    tt.Width =  (750 * (finish - start))
    tt.Name = "tt" & Str$(tt.Top) & Str$(tt.Left)
End Sub
```

```
' Tile method
' Tile the open document windows (.APR files).

CurrentApplication.ApplicationWindow.Tile()
```

```
' TimerInterval Property
' Find out what the timer interval for the current view is set to.
Dim Tmr As Long
Tmr = CurrentApplication.ActiveView.TimerInterval

' Or

' Set the timer interval for the current view to 15 seconds.
CurrentApplication.ActiveView.TimerInterval = 15000
```

```
' Top (Border) property
' This script comes from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application.
' Create a new text block with a 1 point left and right ultramarine border.

Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    t = 1635 & (330 * t)
    Set tt = New textbox(currentview.Body)
    tt.text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $apr1point
    tt.Border.Color.SetRGB (color_ultramarine)
End Sub
```

```
' Top property
' Example from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
' Display the reservation owner in the correct time slot
' in the current view body.
' Called from readBlock
   '  txt                  reservation owner
   '  start                reservation start time
   '  finish               reservation end time
   '  roomName             name or number of reserved room
' * RUNTIME DEPENDENCIES
' * Constants: Uses constants defined by LotusScript defined in
' * LSCONST.LSS.
' * Globals: Uses the global array Rooms() filled by the readBlock
' * sub.

' Declare variables
    Dim tt As textbox    ' New text block to hold the reservation
                                          ' owner's name in the view
    Dim i As Integer     ' Index of array with the room names

' Index of the room that matches the roomName passed in.
' Determine the vertical placement of the reservation in the view.
    Dim matchedRoom As Integer

' Offset and multiplier for the vertical placement of the
' reservation.
    Dim verticalPlacement As Integer

' Search through the global array Rooms to find the room passed
' in from the Schedule database using the sub readBlock.
     ' Set matchedRoom to the index of the room passed in.
    For i = 0 To Ubound(Rooms)
        If Rooms(i) = roomName Then
            matchedRoom = i
            i = Ubound(Rooms)
        End If   ' If element matches the room passed in.
    Next

' Set position and display for the reservation.

' Header in the view takes up 1635 twips, each row in the table
' is 330 twips tall.
    verticalPlacement = 1635 + (330 * matchedRoom)

' Create the text block to hold the reservation.
```

```
    Set tt = New textbox(currentview.body)

' Fill the text block with the reservation owner's name
' and spaces to center the text properly.
    tt.Text = " " + txt + " "

' Set display properties for the text block to match the form.
    tt.Font.Size = 8
      ' Use LotusScript constants for border style.
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
      ' Use Approach constants for line width.
    tt.Border.Width = $apr1point
      ' Use LotusScript constants for color.
    Call tt.Border.Color.SetRGB(COLOR_ULTRAMARINE)
    Call tt.Background.Color.SetRGB (COLOR_50_GRAY)

' Set the position of the text block to correspond to the
' correct room and time.
    tt.Height = 325
    tt.Top = verticalPlacement   ' Current offset from top of
                                                          ' form

' Convert reservation time (passed in) to the horizontal
' location and length on the form.
    tt.Left = (((start - 8) * 750) + 945)
    tt.Width =  (750 * (finish - start))

' Add a prefix to the name of the text block so the clearDisplay
' function can delete the reservation.
    tt.Name = "tt" + Str$(tt.Top) + Str$(tt.Left)

End Sub   ' displayBlock
```

```
' Type property
' Create a list box and fill it with the names of the views available in
' the document (.APR file).
Dim MyListBox As ListBox
Dim MyViewsArray(20) As String
Dim MyNumberViews As Integer

' Check to find out if the current view is a form.
If (CurrentView.Type = $aprForm) Then
    'Create a new list box in the current view.
    Set MyListBox = New ListBox(CurrentView.Body, 1)
    MyNumberViews = CurrentDocument.Views.Count   ' Get the number of views.
    For i = 0 To MyNumberViews-1   ' Fill the array with the view names.
        MyViewsArray(i) = CurrentDocument.Views(i).Name
    Next
    MyListBox.SetList(MyViewsArray)' Fill the list box with the
                                   ' list of view names.
Else
    Messagebox "You must be on a form."
End If
```

```
' UncheckedValue property
' Determine if a check box is checked, and then display a message that
' includes the checked or unchecked value.
' This script is placed in an event script for an object in the same
' view as the check box.
If (Source.ObjCheckBox.IsChecked) Then
    MessageBox("The check box is checked and its value is " &
Source.ObjCheckBox.CheckedValue)
Else
    MessageBox("The check box is not checked and its value is " &
Source.ObjCheckBox.UnCheckedValue)
End If
```

```
' Underline property
' Find out if the text in the LastName field is underlined.
Dim Underln As Integer
Underln = Source.LastName.Font.Underline

' Or

' Set the text in the LastName field to be underlined.
Source.LastName.Font.Underline = True
```

```
' UpdateRow
'This is the modifyRooms global function from the Schedule
'SmartMaster application.

Function modifyRooms
     Dim Con As New Connection
     Dim Qry As New Query
     Dim ReSet As New ResultSet

     modifyRooms = False

     If Con.ConnectTo("dBASE IV") Then
          Set Qry.Connection = Con
          Qry.Tablename = currentdocument.tables(0).path & "rooms"
          Set ReSet.Query = Qry
          If (ReSet.Execute)Then
               If (ReSet.numrows) Then
                    ReSet.firstrow
                    Do
                         ReSet.deleteRow
                    Loop While (ReSet.numrows)
               End If
               modifyRooms = True
               For i = 0 To Ubound(rooms)
                    ReSet.addrow
                    ReSet.setvalue "room", rooms(i)
                    ReSet.updaterow
               Next
          End If
     End If
     con.disconnect

End Function
```

```
'UserID property
'Sets the user ID for a server connection.
Dim C As New Connection
C.UserID = "UserID"
C.Password = "UserPassword"
CkConnectTo = C.ConnectTo("SQL Server", C.UserID, C.Password, "sqlsvr_nt351")
C.Disconnect
```

```
' User property
Sub DocumentReport()
    ' This script prints a report of all of the document information
    ' to the output window.

    ' Print each of the items to the output window.
    Print "Author: " & CurrentDocument.Author
    Print "Description: " & CurrentDocument.Description
    Print "Keywords: " & CurrentDocument.Keywords
    Print "User: " & CurrentDocument.User

    Print "FileName: " & CurrentDocument.Filename
    Print "FullName: " & CurrentDocument.FullName
    Print "Path of the .APR: " & CurrentDocument.Path

    Print "Creation Date: " & CurrentDocument.CreateDate
    Print "LastModified: " & CurrentDocument.LastModified

    If (CurrentDocument.Modified) Then   'If the document has been modified...
        Print "The document has been modified: " & Str(CurrentDocument.NumRevisions)
& " times."
    Else
        Print "The document hasn't been modified."
    End If
    Print "Number of joins: " & Str(CurrentDocument.NumJoins)
    Print "Number of tables in the .APR: " & Str(CurrentDocument.NumTables)
    Print "Number of views in the .APR: " & Str(CurrentDocument.NumViews)
End Sub
```

```
' Value property
' Determine the value of the current check box.
Dim ChkBoxVal as String
ChkBoxVal = Source.ObjCheckBox.Value

' Or
' Set the value of the current check box to 1.
Source.ObjCheckBox.Value = 1
```

```
' VarTable property
' Retrieve the Variable Table for the current document.
Dim Tbl As Table
Set Tbl = CurrentApplication.ActiveDocument.VarTable
```

```
' Views property
' In this script example we create a listbox and fill
' the listbox with the views available.
Dim LstBox As ListBox
Dim aryVws(20) As String
Dim NumVws As Integer

' Check to find out if the current view is a form.
If (CurrentView.Type = $aprForm) Then
    ' Create a new listbox on the current view
    Set LstBox = New ListBox(CurrentView.Body, 1)
    NumVws = CurrentDocument.Views.Count   ' Get the number of views
    For i = 0 To NumVws-1   ' Fill the array with the view names.
        aryVws(i) = CurrentDocument.Views(i).Name
    Next
    LstBox.SetList(aryVws)   ' Set the listbox with the list of views.
Else
    Messagebox "You must be on a form."
End If
```

```
' ViewTabVisible property
Sub CleanScreen()
    ' This script performs the same function
    ' as the Clean Screen menu item on the Edit menu.
    CurrentWindow.Redraw=False   ' Turn off redraw temporarily
    ' Turn off each bar.
    CurrentWindow.ActionBarVisible=False
    CurrentWindow.IconBarVisible=False
    CurrentWindow.StatusBarVisible=False
    CurrentWindow.ViewTabVisible=False
    CurrentWindow.Redraw=True   ' Turn redraw back on
    CurrentWindow.Repaint       ' Now repaint the window.
End Sub
```

```
' Visible property
' Example from the SetDefaultButtonText global function in
' the Checkbook Register SmartMaster application

Sub SetDefaultButtonText
    If currentview.Name="Data Entry Screen" Then
        If currentview.Body.Commitflag.Text="1" Then
            currentview.Body.Voidbutton.Text="Remove this transaction from the
balance"
            startcheck
        Else
            currentview.Body.Voidbutton.Text="Apply this transaction to the balance"
        End If
        If currentview.Body.Transtype.Text="Check" Then
            currentview.Body.Checknumber.Visible=True
            currentview.Body.Depositnumber.Visible=False
        Else
            If currentview.Body.Transtype.Text="Deposit" Then
                currentview.Body.Checknumber.Visible=False
                currentview.Body.Depositnumber.Visible=True
            Else
                currentview.Body.Checknumber.Visible=False
                currentview.Body.Depositnumber.Visible=False
            End If
        End If
    End If
    currentwindow.Repaint
End Sub
```

```
' Width (Border) property
' This script comes from the displayBlock global function
' in the Schedule SmartMaster application.

Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
    Dim tt As textbox

    Dim h As Integer
    Dim i As Integer

    t = 1635 + (330 * t)
    Set tt = New Textbox(currentview.Body)
    tt.Text = " " & txt & " "
    tt.Font.Size = 8
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
    tt.Border.Width = $ItsBorderThick
    tt.Border.Color.SetRGB (color_ultramarine)
    tt.Background.Color.SetRGB (color_50_gray)
    tt.Height = 325
    tt.Top = t
    tt.Left = (((start - 8) * 750) + 960)
    tt.Width =  (750 * (finish - start))
End Sub
```

```
' Width property
' Example from the displayBlock global function
' in the Meeting Room Scheduler SmartMaster application
Sub displayBlock(txt As String, start As Double, finish As Double, roomName As String)
' Display the reservation owner in the correct time slot
' in the current view body.
' Called from readBlock
   '  txt                  reservation owner
   '  start                reservation start time
   '  finish               reservation end time
   '  roomName             name or number of reserved room

' * RUNTIME DEPENDENCIES
' * Constants: Uses constants defined by LotusScript defined in
' * LSCONST.LSS.
' * Globals: Uses the global array Rooms() filled by the readBlock
' * sub.

' Declare variables
    Dim tt As textbox    ' New text block to hold the reservation
                                          ' owner's name in the view
    Dim i As Integer     ' Index of array with the room names

' Index of the room that matches the roomName passed in.
' Determine the vertical placement of the reservation in the view.
    Dim matchedRoom As Integer

' Offset and multiplier for the vertical placement of the
' reservation.
    Dim verticalPlacement As Integer

' Search through the global array Rooms to find the room passed
' in from the Schedule database using the sub readBlock.
     ' Set matchedRoom to the index of the room passed in.
    For i = 0 To Ubound(Rooms)
        If Rooms(i) = roomName Then
            matchedRoom = i
            i = Ubound(Rooms)
        End If   ' If element matches the room passed in.
    Next

' Set position and display for the reservation.

' Header in the view takes up 1635 twips. The height of each row in the table
' is 330 twips.
    verticalPlacement = 1635 + (330 * matchedRoom)
```

```
' Create the text block to hold the reservation.
    Set tt = New textbox(currentview.body)

' Fill the text block with the reservation owner's name
' and spaces to center the text properly.
    tt.Text = " " + txt + " "

' Set display properties for the text block to match the form.
    tt.Font.Size = 8
      ' Use LotusScript constants for border style.
    tt.Border.Style = $ltsBorderStyleNone
    tt.Border.Left = True
    tt.Border.Right = True
    tt.Border.Top = False
    tt.Border.Bottom = False
      ' Use Approach constants for line width.
    tt.Border.Width = $apr1point
      ' Use LotusScript constants for color.
    Call tt.Border.Color.SetRGB(COLOR_ULTRAMARINE)
    Call tt.Background.Color.SetRGB (COLOR_50_GRAY)

' Set the position of the text block to correspond to the
' correct room and time.
    tt.Height = 325
    tt.Top = verticalPlacement   ' Current offset from top of
                                                          ' form

' Convert reservation time (passed in) to the horizontal
' location and length on the form.
    tt.Left = (((start - 8) * 750) + 945)
    tt.Width =  (750 * (finish - start))

' Add a prefix to the name of the text block so the clearDisplay
' function can delete the reservation.
    tt.Name = "tt" + Str$(tt.Top) + Str$(tt.Left)

End Sub   ' displayBlock
```

```
' Windows property
' Retrieve the collection of Windows on the Application.

Dim Wins As Variant
Set Wins = CurrentApplication.Windows
```

```
' Window property
' Turn redraw off while you move and add objects
' to a form, then redraw the entire form when your done.

CurrentView.Window.Redraw = False      ' Turn off redraw off for the current document.
CurrentView.Body.LastName.Left = 1440  ' Move the LastName field.
CurrentView.Body.FirstName.Left = 2880 ' Move the FirstName field.
CurrentView.Body.Address.Left = 1440   ' Move the Address field.
CurrentView.Body.City.Left = 1440      ' Move the City field.
CurrentView.Window.Redraw = True       ' Turn redraw back on.
```

```
' Yellow property
' Find out how much yellow is in the background color of the current object.

Dim b As Long                       ' Create a variable.

b = source.Background.Color.Yellow  ' Determine the amount of yellow.
Print b                             ' Print the amount of yellow.
```

Approach: ActionBarVisible property

{button ,AL(`H_LAS_DOCWINDOW_CLASS',0)} See list of classes

{button ,AL(`H_las_ACTIONBARVISIBLE_EXSCRIPT',1)} See example

Sets or returns whether the action bar is visible.

Data type

Integer

Syntax

docwindowobject.**ActionBarVisible** = *flag*

flag = *docwindowobject*.**ActionBarVisible**

Legal values

Value	Description
TRUE	(Default) Display the action bar.
FALSE	Do not display the action bar.

Usage

Hide the action bar when you do not want users to go to Design, create new records, or do a find.

Approach: ActiveDocument property

{button ,AL(`H_LAS_APPLICATION_CLASS',0)} See list of classes

{button ,AL(`H_las_ACTIVEDOCUMENT_EXSCRIPT',1)} See example

(Read-only) Returns the name of the current document object (.APR file).

Data type

Document

Syntax

Set *documentobject* = *applicationobject*.**ActiveDocument**

Legal values

The current document.

Usage

Determine which document (.APR file) a user is currently viewing. If it is not the one you want them to use, you can use the OpenDocument or Activate methods to switch to a different document.

Approach: ActiveDocWindow property

{button ,AL(`H_LAS_APPLICATION_CLASS',0)} See list of classes

{button ,AL(`H_las_ACTIVEDOCWINDOW_EXSCRIPT',1)} See example

(Read only) Returns the active document window (.APR file).

Data type

DocWindow

Syntax

Set *docwindowobject* = *applicationobject*.**ActiveDocWindow**

Legal values

The active document window.

Usage

Determine which document window is active so you can manipulate records and change the view settings, such as those for displaying a set of icons and other display options.

Approach: ActiveView property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_DOCWINDOW_CLASS',0)} See list of classes

{button ,AL(`H_las_ACTIVEVIEW_EXSCRIPT',1)} See example

Sets or returns the currently active view in the document (.APR file).

Data type

Variant

Syntax

Set *applicationobject*.**ActiveView** = *existingviewobject*

Set *objectvariable* = *applicationobject*.**ActiveView**

or

Set *docwindowobject*.**ActiveView** = *existingviewobject*

Set *objectvariable* = *docwindowobject*.**ActiveView**

Legal values

An existing view object.

Usage

Change to a different view in the document when you want users to perform an action specific to a view.

For example, switch to the view containing the data you want users to review.

Approach: Alignment property

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS',0)} See list of classes

{button ,AL(`H_las_ALIGNMENT_EXSCRIPT',1)} See example

Sets or returns the alignment for fields and summary panels.

- For fields, the Alignment property aligns the text in the element to the left, right, or center.
- For summary panels, the Alignment property aligns the panel with the left edge, right edge, or center of the report.

Data type

Long

Syntax

objectname.**Alignment** = *value*

value = *objectname*.**Alignment**

Legal values

Value	Description
$LtsAlignmentLeft	(Default) Align the text or panel to the left.
$LtsAlignmentHorizCenter	Align the text or panel in the center.
$LtsAlignmentRight	Align the text or panel to the right.

Usage

Align the text for a new display element, or align a new summary panel in a report.

Approach: AllowDrawing property

{button ,AL(`H_LAS_PICTUREPLUS_CLASS',0)} See list of classes

{button ,AL(`H_las_ALLOWDRAWING_EXSCRIPT',1)} See example

Sets or returns if you can draw with the mouse in PicturePlus fields.

Data type

Integer

Syntax

pictureplusobject.**AllowDrawing** = *flag*

flag = *pictureplusobject*.**AllowDrawing**

Legal values

Value	Description
FALSE	(Default) Users cannot draw in the PicturePlus field.
TRUE	Users can draw in the PicturePlus field.

Approach: AlternateColors property

{button ,AL(`H_LAS_REPEATINGPANEL_CLASS',0)} See list of classes

Sets or returns whether the lines in a repeating panel are set to alternate colors. The alternating colors used are the background color of the repeating panel and the background color of the form.

Data type

Integer

Syntax

repeatingpanel.**AlternateColors** = *flag*

flag = *repeatingpanel*.**AlternateColors**

Legal values

Value	Description
FALSE	(Default) Do not display alternate colors.
TRUE	Display alternate colors.

Usage

Display a repeating panel in alternate colors when you want to distinguish between the data in each row.

Approach: ApplicationWindow property

{button ,AL(`H_LAS_APPLICATION_CLASS',0)} See list of classes

{button ,AL(`H_las_APPLICATIONWINDOW_EXSCRIPT',1)} See example

(Read-only) Returns the application window.

Data type

ApplicationWindow

Syntax

Set *applicationwindowobject* = *applicationobject*.**ApplicationWindow**

Legal values

The active application window.

Usage

Determine which application window is active so you can change the window settings or execute a menu command.

Approach: Application property (ApplicationWindow class)

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS',0)} See list of classes

(Read-only) Returns the Application object to which the window belongs.

Data type

Application

Syntax

Set *applicationobject* = *applicationwindowobject*.**Application**

Legal Values

Any Approach application.

Usage

If users are running several instances of Approach, you can determine which instance they are currently using, so you can change the application-wide settings.

Approach: Application property (Application class)

{button ,AL(`H_LAS_APPLICATION_CLASS',0)} See list of classes

{button ,AL(`H_las_APPLICATION_EXSCRIPT',1)} See example

(Read-only) Returns the instance of the application itself.

Data type

Application

Syntax

Set *applicationobject* = *applicationobject***.Application**

Legal Values

The current active application.

Usage

Determine which application is active so you can determine application settings, such as its path and which documents (.APR files) are open.

Approach: ApplyFoundSet property

{button ,AL(`H_LAS_CROSSTAB_CLASS',0)} See list of classes

Applies the current found set to a crosstab.

Data type

Integer

Syntax

crosstabobject.**ApplyFoundSet** = *flag*

flag = *crosstabobject*.**ApplyFoundSet**

Legal values

Value	Description
TRUE	(Default) Apply the found set to a crosstab.
FALSE	Do not apply the found set to a crosstab.

Approach: Author property

{button ,AL(`H_LAS_DOCUMENT_CLASS',0)} See list of classes

{button ,AL(`H_las_AUTHOR_EXSCRIPT',1)} See example

(Read-only) Returns the name of the person who created the document, or the name of the person who last entered their name as the author in the Approach File Properties dialog box.

Data type

String

Syntax

name = *documentobject*.**Author**

Legal values

Any string.

The default value for the Author property is the name of the person who created the document.

Approach: Background property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_BACKGROUND_EXSCRIPT',1)} See example

Sets or returns the background object for the current display element or panel.

Data type

Background

Syntax

set *displayelementobject*.**Background** = *backgroundobject*

set *backgroundobject* = *displayelementobject*.**Background**

Legal values

For panels, you can set this property to the background of another view. The panel will pick up the background color of the other panel.

You can set this property to any color object as listed in the Color class.

Approach: Baseline property

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

{button ,AL(`H_las_BASELINE_EXSCRIPT',1)} See example

Displays a dashed line in a field for entering text.

Data type

Integer

Syntax

displayelementobject.**Border.Baseline** = *flag*

flag = *displayelementobject*.**Border.Baseline**

or

borderobject.**Baseline** = *flag*

flag = *borderobject*.**Baseline**

Legal values

Value	Description
FALSE	(Default) Do not display the baseline.
TRUE	Display the baseline.

Usage

Display a baseline in a field box to indicate where users can enter text.

You can use a baseline without any other borders. Data entered by users sits on the dashed line.

Approach: Black property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_BLACK_EXSCRIPT',1)} See example

(Read-only) Returns the black component of a CMYK value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**Black**

Legal values

Any integer from 0 to 255.

Usage

0 represents no black and 255 represents the highest amount of black available.

Approach: Blue property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_BLUE_EXSCRIPT',1)} See example

(Read-only) Returns the blue component of an RGB value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**Blue**

Legal values

Any integer from 0 to 255.

Usage

0 represents no blue and 255 represents the highest amount of blue available.

Use the SetRGB method to change the settings of the Red, Green, and Blue properties all at once.

Approach: Bold property

{button ,AL(`H_LAS_FONT_CLASS;',0)} See list of classes

Sets or returns if the text of the current display element is bold type.

Data type

Integer

Syntax

displayelementobject.**Font.Bold** = *flag*

flag = *displayelementobject*.**Font.Bold**

or

fontobject.**Bold** = *flag*

flag = *fontobject*.**Bold**

Legal values

Value	Description
FALSE	(Default) The font is not bold.
TRUE	The font is bold.

Approach: Border property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_BORDER_EXSCRIPT',1)} See example

Sets or returns the border object of the current display element or panel.

Data type

Border

Syntax

set *displayelementobject*.**Border** = *borderobject*

set *borderobject*= *displayelementobject*.**Border**

Legal values

Any border object as listed in the Border class.

Usage

Change the border of a new display element or panel so it is consistent with other display elements or panels.

For example, change a new field box to have a blue border.

Approach: Bottom property

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

{button ,AL(`H_las_BOTTOM_BORDER_EXSCRIPT',1)} See example

Sets or returns whether the bottom border of a display element is displayed.

Data type

Integer

Syntax

borderobject.**Bottom** = *flag*

flag = *borderobject*.**Bottom**

Legal values

Value	Description
FALSE	(Default) Do not display the bottom border.
TRUE	Display the bottom border.

Usage

The border of a display element is transparent until you set its color using the SetRGB method.

Approach: CheckedValue property

{button ,AL(`H_LAS_CHECKBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_CHECKEDVALUE_EXSCRIPT',1)} See example

Sets or returns the value associated with checked state of a check box.

Note This value does not reflect the current state of the check box.

Data type

String

Syntax

*checkboxobject***.CheckedValue** = *value*

value = *checkboxobject***.CheckedValue**

Legal values

Default value: Yes

You can set this property to any string up to 256 characters.

Usage

Determine the value of a checked check box, regardless of its current state.

For example, you can display a set of radio buttons below only those check boxes that would have a checked value of Yes. The radio buttons are displayed whether or not the check boxes are checked.

Approach: ClickedValue property

{button ,AL(`H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_CLICKEDVALUE_EXSCRIPT',1)} See example

Sets or returns the value associated with the clicked state of a radio button .

Data type

String

Syntax

radiobuttonobject.**ClickedValue** = *value*

value = *radiobuttonobject*.**ClickedValue**

Legal values

Any string up to 256 characters.

Approach: Color property

{button ,AL(`H_LAS_BACKGROUND_CLASS;H_LAS_BORDER_CLASS;H_LAS_FONT_CLASS;H_LAS_LINESTYLE_CLASS;',0)} See list of classes

Specifies the color object for the border, font, linestyle or background of a display element.

Data type

Color

Syntax

Object	Syntax
Background	*displaylementobject*.**Background.Color** = *colorobject*
	colorobject = *displayelemetobject*.**Background.Color**
Border	*displayelementobject*.**Border.Color** = *colorobject*
	colorobject = *displayelementobject*.**Border.Color**
Font	*displayelementobject*.**Font.Color** = *colorobject*
	colorobject = *displayelementobject*.**Font.Color**
Linestyle	*displayelementobject*.**Linestyle.Color** = *colorobject*
	colorobject = *displayelementobject*.**Linestyle.Color**

Legal values

The default color for this property is white.

You can set this property to any color object as listed in the Color class.

Approach: Connection property

{button ,AL(`H_LAS_QUERY_CLASS;',0)} See list of classes

{button ,AL(`H_las_CONNECTION_EXSCRIPT',1)} See example

Sets or returns the Connection object used in the query.

Data type

Connection

Syntax

Set *queryobject*.**Connection** = *connectionobject*

Set *connectionobject* = *queryobject*.**Connection**

Legal values

Any existing Connection object.

Usage

You must set this property to a Connection object to execute a query or produce a result set.

The connection must be open when you call the Execute method.

Approach: Count property

{button ,AL(`H_LAS_BASECOLLECTION_CLASS;H_LAS_COLLECTION_CLASS;H_LAS_LISTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_COUNT_EXSCRIPT',1)} See example

(Read-only) Returns the number of objects in a BaseCollection, Collection, or ListBox object.

Data type

Integer

Syntax

value = *basecollectionobject*.**Count**

or

value = *collectionobject*.**Count**

or

value = *listboxobject*.**Count**

Legal values

The number of objects in the BaseCollection, Collection, or ListBox object.

Approach: CurrentPageNum property

{button ,AL(`H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;',0)} See list of classes

{button ,AL(`H_las_CURRENTPAGENUM_EXSCRIPT',1)} See example

Sets or returns the current page number of a form or form letter.

Data type

Integer

Syntax

formobject.**CurrentPageNum** = *value*

value = *formobject*.**CurrentPageNum**

or

formletterobject.**CurrentPageNum** = *value*

value = *formletterobject*.**CurrentPageNum**

Legal values

Any number between 1 and the number of pages on the form or form letter. To determine the number of pages in the form or form letter, use the NumPages property.

The default value is the current page number of the form or form letter.

Approach: CurrentRow property

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_CURRENTROW_EXSCRIPT',1)} See example

Sets or returns the current row (record) in a result set.

Data type

Long

Syntax

resultsetobject.**CurrentRow** = *value*

value = *resultsetobject*.**CurrentRow**

Legal values

Any number between 1 and the number of rows in the result set. Determine the number of rows using the NumRows method. A value of zero indicates that there are no records in the result set.

Usage

Setting the current row prepares Approach to read or edit data from that row of the result set.

Approach: Cyan property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_CYAN_EXSCRIPT',1)} See example

(Read-only) Returns the cyan component of a CMYK value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**CYAN**

Legal values

Any integer from 0 to 255.

Usage

0 represents no cyan and 255 represents the highest amount of cyan available.

Approach: DataField property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_DATAFIELD_EXSCRIPT',1)} See example

Sets or returns the name of the field in the table (database file) bound to the display element in the view, such as a field box or PicturePlus object.

Data type

String

Syntax

displayelement object.**DataField** = *tablefield*

tablefield = *displayelementobject*.**DataField**

Legal values

Any field in the table.

Usage

After you create a new display element in a view, such as a field box, set this property to the table field you want the display element to represent.

Approach: DataSourceName property

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

(Read-only) Returns the name of the data source type specified in the Connection object.

Data type

String

Syntax

string = *connectionobject*.**DataSourceName**

Legal values

Any data source type that Approach can read using built-in drivers or a driver you supply.

Use the ListDataSources method to produce a list of the available data source types.

Approach: DataTable property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS',0)} See list of classes

{button ,AL(`H_las_DATATABLE_EXSCRIPT',1)} See example

Sets or returns the name of the table (database file) bound to the display element in the view, such as a field box or PicturePlus object.

Data type

String

Syntax

displayelementobject.**DataTable** = *tablename*

tablename = *displayelementobject*.**DataTable**

Legal values

Any of the tables associated with the document (.APR file).

Usage

After you create a new display element, such as a field box, set this property to the table that contains the field you want the display element to represent.

Approach: Documents property

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

{button ,AL(`H_las_DOCUMENTS_EXSCRIPT',1)} See example

(Read-only) Returns a list of the documents (.APR files) open in an Approach application (.EXE file).

Data type

BaseCollection

Syntax

Set *collectionobject* = *applicationobject*.**Documents**

Usage

This property identifies the open document(s) without having to know them by name.

Identify a single document by specifying its index in the Documents BaseCollection.

For example, if you want to close the first document opened in the current Approach session, specify the first document in Documents (the BaseCollection is numbered starting from zero).

Approach: Document property

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_DOCUMENT_EXSCRIPT',1)} See example

(Read-only) Returns the document object (.APR file) displayed in the current window.

Data type

Document

Syntax

Set *document* = *docwindowobject*.**Document**

Legal values

Any document (.APR file).

Usage

If users have several documents open, determine which document they are currently viewing.

Approach: DrillDownView property

{button ,AL(`H_LAS_CROSSTAB_CLASS;',0)} See list of classes

{button ,AL(`H_las_DRILLDOWNVIEW_EXSCRIPT',1)} See example

Sets or returns the view that shows in detail the data summarized by the crosstab.

Data type

Variant

Syntax

crosstabobject.**DrillDownView** = *viewobject*

viewobject = *crosstabobject*.**DrillDownView**

Legal values

Any view in the document (.APR file).

To set a view as the drill-down view, supply the name of a view object. To unset the property, set it to zero.

Approach: Editable property

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;',0)} See list of classes

Sets or returns whether users can edit text or add a new item in a drop-down box.

Data type

Integer

Syntax

ddboxname.**Editable** = *flag*

flag = *ddboxname*.**Editable**

Legal values

Value	Description
TRUE	(Default) You can edit text or create a new item in the drop-down box.
FALSE	You cannot edit text in the drop-down box; you can only select an item from the list.

Usage

Enter all the valid values when you create a new drop-down box, and then change its property so users cannot add any other values.

Approach: Enabled property

{button ,AL(`H_LAS_BUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_ENABLED_EXSCRIPT',1)} See example

Sets or returns if a button can be clicked. A button is dimmed when it is disabled.

Data type

Integer

Syntax

buttonobject.**Enabled** = *flag*

flag = *buttonobject*.**Enabled**

Legal values

Value	Description
TRUE	(Default) The button is enabled.
FALSE	The button is disabled; it appears dimmed.

Usage

Disable buttons when you do not want users to run the attached macro or script. For example, you can choose not to enable a Close button until users complete the dialog box.

Approach: EncloseLabel property

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

Sets or returns whether to enclose a label within the border of a field.

Data type

Integer

Syntax

borderobject.**EncloseLabel** = *flag*

flag = *borderobject*.**EncloseLabel**

Legal values

Value	Description
FALSE	(Default) Do not enclose the label within the border.
TRUE	Enclose the label within the border.

Approach: Expand property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

{button ,AL(`H_las_EXPAND_EXSCRIPT',1)} See example

Sets or returns whether a body panel, summary panel, or field box expands when printing.

Data type

Integer

Syntax

object.**Expand** = *flag*

flag = *object*.**Expand**

Legal values

Value	Description
TRUE	(Default) Expand the display element when printing.
FALSE	Do not expand the display element when printing.

Usage

Expand a field or panel boundary if it contains more data than it can display.

For example, if a description field displays only 20 characters in the view, but contains more characters, expand the field so users can see all the text when it is printed.

Approach: FontName property

{button ,AL(`H_LAS_FONT_CLASS',0)} See list of classes

{button ,AL(`H_las_FONTNAME_EXSCRIPT',1)} See example

Sets or returns the name of the font for the current display element.

Data type

String

Syntax

fontobject.**FontName** = *string*

string = *fontobject*.**FontName**

or

displayelementobject.**Font.FontName** = *string*

string = *displayelemetobject*.**Font.FontName**

Legal values

The default font is Arial. Any fonts currently installed on the computer are also legal values.

Usage

Change the font when you add new display elements and you want to make them consistent with other interface elements. For example, you can change the font of a text block showing the title of a form from Arial to Times New Roman.

Approach: Font property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_FONT_EXSCRIPT',1)} See example

Sets or returns the font object used to define the font attributes of display elements such as buttons and field boxes.

Data type

Font

Syntax

displayelementobject.**Font** = *fontobject*

fontobject = *displayelementobject*.**Font**

Legal values

Any font object as listed in the Font class.

Usage

Change the font of the display element to make it consistent with, or more noticeable than the other display elements in a view. For example, you can set the text of an important button to blue, bold, 10-point type.

Approach: Green property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_GREEN_EXSCRIPT',1)} See example

(Read-only) Returns the green component of an RGB value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**Green**

Legal values

0 represents no green and 255 represents the highest amount of green available.

Usage

Use the SetRGB method to change the settings of the Red, Green, and Blue properties all at once.

Approach: Height property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_HEIGHT_EXSCRIPT',1)} See example

Sets or returns the height, in twips, of a display element or panel.

Data type

Long

Syntax

object.**Height** = value

value = *object*.**Height**

Legal values

The default value for HeaderFooterPanel is 0. You must change this to show a header or footer.

The maximum height of the display element or panel is the maximum page size allowed by your page setup.

Approach: HideMargins property

{button ,AL(`H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;',0)} See list of classes

{button ,AL(`H_las_HIDEMARGINS_EXSCRIPT',1)} See example

Sets or returns whether the margins are displayed.

Data type

Integer

Syntax

viewobject.**HideMargins** = *flag*

flag = *viewobject*.**HideMargins**

Legal values

Value	Description
FALSE	(Default) Do not hide the margins.
TRUE	Hide the margins.

Approach: IsBeginOfData property

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

(Read-only) Returns whether the current row is the first row in the result set.

Data type

Integer

Syntax

integer = *resultsetobject*.**IsBeginOfData**

Legal values

Value	Description
TRUE	The current row is the first row of the result set.
FALSE	The current row is not the first row of the result set.

Approach: IsChecked property

{button ,AL(`H_LAS_CHECKBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_ISCHECKED_EXSCRIPT',1)} See example

(Read-only) Returns the state of a check box.

Note The SetState method sets the initial value of the check box to be checked or unchecked.

Data type

Integer

Syntax

flag = checkboxobject.**IsChecked**

Legal values

Value	Description
FALSE	The check box is not checked.
TRUE	The check box is checked.

Usage

Determine if a check box is checked, and then perform another task based on the result. For example, if users turn on the "Non smoking" check box, display only the list of available rooms that are designated "Non smoking".

Approach: IsConnected property

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

(Read-only) Returns the data connection status.

Data type

Integer

Syntax

integer = *connectionobject*.**IsConnected**

Legal values

Value	Description
TRUE	The connection is active.
FALSE	The connection is not active.

Usage

Use this property to verify that your data connection is active to avoid sending error messages to users when your script attempts an operation that requires a connection.

Approach: IsEndOfData property

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

(Read-only) Returns whether the current row is the last row in the result set.

Data type

Integer

Syntax

integer = *resultsetobject*.**IsEndOfData**

Legal values

Value	Description
TRUE	The current row is the last row in the result set.
FALSE	The current row is not the last row in the result set.

Approach: IsReadOnly property

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

(Read-only) Returns whether a result set is read-only or read-write.

You cannot add, delete, or change records in a read-only result set.

Data type

Integer

Syntax

integer = *resultsetobject*.**IsReadOnly**

Legal values

Value	Description
TRUE	The result set is read-only.
FALSE	The result set is read-write.

Approach: IsResultSetAvailable property

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

(Read-only) Returns whether the result set is available.

Data type

Integer

Syntax

integer = *resultsetobject*.**IsResultSetAvailable**

Legal values

Value	Description
TRUE	The result set is available.
FALSE	The result set is not available.

Usage

This property determines whether the source table for the result set still exists and is connected to Approach. For example, check the value of IsResultSetAvailable before calling the Transactions or UpdateRow methods to make sure that the edit of the result set will be successful.

Approach: Italic property

{button ,AL(`H_LAS_FONT_CLASS;',0)} See list of classes

{button ,AL(`H_las_ITALIC_EXSCRIPT',1)} See example

Sets or returns if the text of a display element is italicized.

Data type

Integer

Syntax

fontobject.**Italic** = *flag*

flag = *fontobject*.**Italic**

or

displaylementobject.**Font.Italic**= *flag*

flag = *displayelemetobject*.**Font.Italic**

Legal values

Value	Description
FALSE	(Default) The font is not italic.
TRUE	The font is italic.

Approach: LabelAlignment property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_LABELALIGNMENT_EXSCRIPT',1)} See example

Sets or returns the alignment of a label with its field in the view.

Data type

Long

Syntax

displayelementobject.**LabelAlignment** = *value*

value = *displayelementobject*.**LabelAlignment**

Legal values

Value	Description
$ltsAlignmentLeft	(Default) Aligns the label with the left edge of the display element.
$ltsAlignmentHorizCenter	Aligns the label with the center of the display element.
$ltsAlignmentRight	Aligns the label with the right edge of the display element.

Usage

After you create a new field in a view, change the label alignment to a different position from the default.

Approach: LabelFont property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_LABELFONT_EXSCRIPT',1)} See example

Sets or returns the font for the label of a field in a view.

Data type

Font

Syntax

Set *displayelementobject*.**LabelFont** = *value*

Set *value* = *displayelementobject*.**LabelFont**

Legal values

Any font object as listed in the Font class.

Usage

After you create a new field in a view, you can change the label to a different font, for example Times New Roman.

Approach: LabelPosition property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_LABELPOSITION_EXSCRIPT',1)} See example

Sets or returns the position of the label relative to its associated field.

Data type

Long

Syntax

displayelementobject.**LabelPosition** = *value*

value = *displayelementobject*.**LabelPosition**

Legal values

Value	Description: The label appears
$LtsPositionTop	(Default)Above the display element.
$LtsPositionBottom	Below the display element.
$LtsPositionLeft	To the left of the display element.
$LtsPositionRight	To the right of the display element.
$LtsPositionNone	The display element does not have a label.

Usage

After you create a new field in a view, change the label to appear in a different position from the default.

Approach: LabelText property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_LABELTEXT_EXSCRIPT',1)} See example

Sets or returns the text for the label of a field in a view.

Data type

String

Syntax

displayelementobject.**LabelText** = *stringexp*

stringexp = *displayelementobject*.**LabelText**

Legal values

Any string up to 256 characters.

Usage

The name of a new field in a view and its label are the same unless you change the text of the label. You may want a more descriptive or readable label for the field in the view.

For example, if the name of a radio button is FLD_RESERVE_CORP_RATE, you can change its label to Corporate Rate.

Approach: Left property (Border class)

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

{button ,AL(`H_las_LEFT_BORDER_EXSCRIPT',1)} See example

Sets or returns whether to display a border along the left side of a display element.

Data type

Integer

Syntax

borderobject.**Left** = *flag*

flag = *borderobject*.**Left**

Legal values

Value	Description
FALSE	(Default) Do not display the left border.
TRUE	Display the left border.

Usage

The border of a display element is transparent until you set its color using the SetRGB method.

Approach: Left property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_LEFT_EXSCRIPT',1)} See example

Sets or returns the distance, measured in twips, between the left edge of a display element or panel and the left edge of its parent.

Data type

Long

Syntax

displayelementobject.**Left** = *value*

value = *displayelementobject*.**Left**

Legal values

If the Left property is a negative value, the display element or panel may be hidden.

Approach: LineSpacing property

{button ,AL(`H_LAS_TEXTBOX_CLASS;',0)} See list of classes

Sets or returns the amount of space between lines of text in a text block.

Data type

Long

Syntax

textboxobject.**LineSpacing** = *value*

value = *textboxobject*.**LineSpacing**

Legal values

Value	Description
$LtsLineSpacingSingle	(Default) Use single-line spacing.
$LtsLineSpacingSingleAndHalf	Use a line-and-a-half spacing.
$LtsLineSpacingDouble	Use two-line spacing.

Usage

Use this property to increase or decrease the amount of space between the lines of text in a text block.

For example, if single line spacing is too hard to read, you can increase it by another half or full line.

Approach: LineStyle property

{button ,AL(`H_LAS_LINEOBJECT_CLASS;',0)} See list of classes

{button ,AL(`H_las_LINESTYLE_EXSCRIPT',1)} See example

Sets and returns the LineStyle object which specifies the style of a line; for example, the line color and pattern.

Data type

LineStyle

Syntax

Set *lineobject*.**LineStyle** = *linestyleobject*

Set *linestyleobject* = *lineobject*.**LineStyle**

Legal Values

Any LineStyle object.

Approach: MacroClick property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_MACROCLICK_EXSCRIPT',1)} See example

Sets or returns the name of the macro to run when users click certain display elements, such as buttons, ellipses, pictures, and so on.

Data type

String

Syntax

displayelementobject.**MacroClick** = *macroname*

macroname = *displayelementobject*.**MacroClick**

Legal values

A string expression whose value is the name of a macro in the document (.APR file).

Approach: MacroDataChange property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_MACRODATACHANGE_EXSCRIPT',1)} See example

Sets or returns the macro to run when the data in a field changes.

Data type

String

Syntax

displayelementobject.**MacroDataChange** = *macroname*

macroname = *displayelementobject*.**MacroDataChange**

Legal values

Any macro in the document (.APR file).

Usage

You can run a macro when users update information in a field. For example, when a user turns on a check box or radio button, run a macro to display an appropriate report.

Approach: MacroTabIn property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_MACROTABIN_EXSCRIPT',1)} See example

Sets or returns the name of the macro to run when the user tabs into a display element.

Data type

String

Syntax

displayelementobject.**MacroTabIn** = *macroname*

macroname = *objectname*.**MacroTabIn**

Legal values

A string expression whose value is the name of a macro in the document (.APR file).

Approach: MacroTabOut property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_MACROTABOUT_EXSCRIPT',1)} See example

Sets or returns the name of the macro to run when the user tabs out of a display element.

Data type

String

Syntax

displayelementobject.**MacroTabOut** = *macroname*

macroname = *displayelementobject*.**MacroTabOut**

Legal values

A string expression whose value is the name of a macro in the document (.APR file).

Approach: Magenta property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_MAGENTA_EXSCRIPT',1)} See example

(Read-only) Returns the magenta component of a CMYK value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**Magenta**

Legal values

Any integer from 0 to 255.

Usage

0 represents no magenta and 255 represents the highest amount of magenta available.

Approach: NamedStyle property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NAMEDSTYLE_EXSCRIPT',1)} See example

Sets or returns a named style for a display element or panel.

Data type

String

Syntax

objectname.**NamedStyle** = *namedstyle*

namedstyle = *objectname*.**NamedStyle**

Legal values

A string expression whose value is a named style in the document (.APR file).

Approach: Name property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHARTVIEW_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DOCUMENT_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_REPORT_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_NAME_EXSCRIPT',1)} See example

Sets or returns the name of a document (.APR file), display element, panel, view, or application.

Data type

String

Syntax

object.**Name** = *namestring*

namestring = *object*.**Name**

Legal values

A string whose value is the name of a document, display element, panel, view, or application.

All display elements on a panel, panels in a view, views in a document, and documents in a directory must have unique names.

Usage

The name of an object is the identifier for the object that appears in the Object drop-down box in the Script Editor.

The Name property is read-only for Document objects.

Approach: NonPrinting property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NONPRINTING_EXSCRIPT',1)} See example

Sets or returns if a display element will print.

For example, you can prevent buttons from printing.

Data type

Integer

Syntax

displayelementobject.**NonPrinting** = *flag*

flag = *displayelementobject*.**NonPrinting**

Legal values

Value	Description
FALSE	(Default) Print the display element.
TRUE	Do not print the display element.

Approach: Orientation property

{button ,AL(`H_LAS_LINEOBJECT_CLASS;',0)} See list of classes

{button ,AL(`H_las_ORIENTATION_EXSCRIPT',1)} See example

Sets or returns the orientation of the line graphic.

Data type

Long

Syntax

lineobject.**Orientation =** *long*

long **=** *lineobject*.**Orientation**

Legal values

Value	Description
$LtsOrientationPosSlope	(Default) The line has a positive slope.
$LtsOrientationNegSlope	The line has a negative slope.

Usage

Slope is measured as the change in length of the line on the y-axis over the change in length on the x-axis. The computer screen's coordinate system starts at the top left corner, and values on the y-axis increase as you move down the screen.

Approach: Page property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_PAGE_EXSCRIPT',1)} See example

(Read-only) Returns the page number on which a display element or panel appears on a form or report.

Data type

Integer

Syntax

value = *displayelementobject*.**Page**

Legal values

For forms, any integer 1 - 5.

For reports, any integer up to the number of pages in the report.

Approach: Parent property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_APPLICATIONWINDOW_CLASS;H_LAS_BODYPANEL_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHARTVIEW_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DOCUMENT_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_FORM_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTURE_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_REPORT_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TABLE_CLASS;H_LAS_TEXTBOX_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;H_LAS_MAILINGLABELS_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_PARENT_EXSCRIPT',1)} See example

(Read-only) Returns the parent object.

Data type

If the object is	Then the Parent data type is
Application, ApplicationWindow or Document	Application
DocWindow, View, Table or any Approach view	Document
A panel	Variant (a view)
A display element	Variant (a panel)

Syntax

Set *parentobject* **=** *object*.**Parent**

Legal values

An existing object.

Usage

For panel and display element objects, declare the object variable for the parent object as Variant or use the abstract classes Panel and View.

Determine the parent application that contains the current application window or document, so you can change the Approach application's settings.

Determine the path of the Approach executable running the current document.

Approach: Pattern property

{button ,AL(`H_LAS_BORDER_CLASS;H_LAS_LINESTYLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_PATTERN_EXSCRIPT',1)} See example

Sets or returns the style used for lines and display element borders.

Data type

Long

Syntax

displayelementobject.**Linestyle.Pattern** = *value*

value = *displayelementobject*.**Linestyle.Pattern**

or

displayelementobject.**Border.Pattern** = *value*

value = *displayelementobject*.**Border.Pattern**

Legal values

For lines, use the following values:

Value	Description
$LtsLineStyleSolid	(Default) Display a solid line.
$LtsLineStyleDouble	Display a double solid line.
$LtsLineStyleDot	Display a dotted line.
$LtsLineStyleLongDash	Display a long dashed line.
$LtsLineStyleMediumDash	Display a medium dashed line.
$LtsLineStyleMediumShortDash	Display a medium and short dashed line.
$LtsLineStyleMediumShortShortDash	Display a medium and two short dashed line.
$LtsLineStyleShortDash	Display a short dashed line.
$LtsLineStyleDashDot	Display an alternating dashed and dotted line.
$LtsLineStyleDashDotDot	Display an alternating dashed and two dots line.
$LtsLineStyleNone	Do not display a line.

For borders, use the following values:

Value	Description
$LtsBorderPatternSolid	(Default) Display a solid line.
$LtsBorderPatternDouble	Display a double solid line.
$LtsBorderPatternDot	Display a dotted line.
$LtsBorderPatternDashed	Display a dashed line.
$LtsBorderPatternDashDot	Display an alternating dashed and dotted line.
$LtsBorderPatternDashDotDot	Display an alternating dashed and two dots line.
$LtsBorderPatternLongDash	Display a long dashed line.
$LtsBorderPatternLowered	Display a lowered three-dimensional effect line.
$LtsBorderPatternNone	Display no border.
$LtsBorderPatternRaised	Display a raised three-

dimensional effect line.

Usage

Change the style of a line or border to indicate a difference in data.

For example, change the border of a field box from dashed to solid when it is a read-only field.

You can also use a line to divide a form into different areas that group fields containing related data.

Approach: Position property

{button ,AL(`H_LAS_PICTUREPLUS_CLASS;',0)} See list of classes

Sets or returns the position of a picture in a PicturePlus field relative to its frame. The frame is divided into a 3 x 3 grid, providing nine possible positions for the picture.

Data type

Long

Syntax

pictureplusobject.**Position** = *value*

value = *pictureplusobject*.**Position**

Legal values

Value	Description
$LtsPositionTopLeft	(Default) The picture is in the top left position (1 in the 9 block grid).
$LtsPositionTop	The picture is in the top center position (2 in the 9 block grid).
$LtsPositionTopRight	The picture is in the top right position (3 in the 9 block grid).
$LtsPositionLeft	The picture is in the middle left position (4 in the 9 block grid).
$LtsPositionCenter	The picture is in the center position (5 in the 9 block grid).
$LtsPositionRight	The picture is in the middle right position (6 in the 9 block grid).
$LtsPositionBottomLeft	The picture is in the bottom left position (7 in the 9 block grid).
$LtsPositionBottom	The picture is in the bottom center position (8 in the 9 block grid).
$LtsPositionBottomRight	The picture is in the bottom right position (9 in the 9 block grid).

Approach: Query property

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_QUERY_EXSCRIPT',1)} See example

Sets or returns the query used to create the result set.

Data type

Query

Syntax

Set *resultsetobject*.**Query** = *queryobject*

Set *queryobject* = *resultsetobject*.**Query**

Legal Values

Any existing query object.

Usage

You must set this property to a Query object to execute the ResultSet object.

Approach: ReadOnly property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_READONLY_EXSCRIPT',1)} See example

Sets or returns if a field is read-only.

Data type

Integer

Syntax

objectname.**ReadOnly** = *flag*

flag = *objectname*.**ReadOnly**

Legal values

Value	Description
FALSE	(Default) The object is read-write.
TRUE	The object is read-only.

Usage

If you want users to view the data in a field but not make any changes to the data, make the field read-only. For example, users cannot change a read-only check box selection.

If you later want users to be able to update the information, you can change the field back to be read-write.

Approach: Reduce property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

{button ,AL(`H_las_REDUCE_EXSCRIPT',1)} See example

Sets or returns whether to decrease the size of a body panel, summary panel, or field box when printing.

Data type

Integer

Syntax

displayelementobject.**Reduce** = *flag*

flag = *displayelementobject*.**Reduce**

Legal values

Value	Description
FALSE	(Default) Do not reduce the display element when printing.
TRUE	Reduce the display element when printing.

Usage

Reduce the size of a field box or panel if it contains a lot of blank space.

For example, if a description field can display 100 characters in the view, but contains only 20 characters, reduce the size of the field box so it does not waste space when it is printed.

If the field box or panel is too small to display all the data, use the Expand property.

Approach: Red property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_RED_EXSCRIPT',1)} See example

(Read-only) Returns the red component of an RGB value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**Red**

Legal values

Any integer from 0 to 255.

Usage

0 represents no red and 255 represents the highest amount of red available.

Approach: Relief property

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_FONT_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_RELIEF_EXSCRIPT',1)} See example

Sets or returns the style of a check box or radio button, or the appearance of field data, a field label or a text block.

Data type

Long

Syntax

displayelementobject.**Relief** = *value*

value = *displayelementobject*.**Relief**

or

displaylementobject.**Font.Relief** = *value*

value = *displayelemetobject*.**Font.Relief**

Legal values

Value	Description
$LtsReliefNone	(Default) The check box, radio button, field data, field label or text block appears normally.
$LtsReliefRaised	The check box, radio button, field data, field label or text block has a 3D raised effect.
$LtsReliefLowered	The check box, radio button, field data, field label or text block has a 3D, engraved effect.

Usage

Change the appearance of a display element to create an attractive, consistent user interface. For example, if you add a new radio button to a view, and existing radio buttons on the view have a 3D, raised effect, change the new radio button to appear the same.

Approach: Right property (Border class)

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

{button ,AL(`H_las_RIGHT_BORDER_EXSCRIPT',1)} See example

Sets or returns whether to display a border along the right side of a display element.

Data type

Integer

Syntax

borderobject.**Right** = *flag*

flag = *borderobject*.**Right**

Legal values

Value	Description
TRUE	(Default) Display the right border.
FALSE	Do not display the right border.

Approach: ShadowColor property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_PICTURE_CLASS;H_LAS_PICTUREPLUS_CLASS;;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SHADOWCOLOR_EXSCRIPT',1)} See example

Sets or returns the color for the shadow of a display element.

Data type

Color

Syntax

displayelementobject.**ShadowColor** = *color*

color = *displayelementobject*.**ShadowColor**

Legal values

The default color for the ShadowColor property is transparent--that is, the display element shows no shadow.

You can set this property to any color object as listed in the Color class.

Approach: ShowArrow property

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SHOWARROW_EXSCRIPT',1)} See example

Sets or returns whether to display an arrow in a drop-down box.

Data type

Integer

Syntax

ddboxname.**ShowArrow** = *flag*

flag = *ddboxname*.**ShowArrow**

Legal values

Value	Description
TRUE	(Default) Show the drop-down arrow.
FALSE	Do not show the drop-down arrow.

Usage

The arrow indicates that the box expands to show more items.

Approach: ShowAsDialog property

{button ,AL(`',0)} See list of classes

{button ,AL(`H_las_SHOWASDIALOG_EXSCRIPT',1)} See example

Sets or returns whether to display the form as a dialog box.

Data type

Integer

Syntax

formobject.**ShowAsDialog** = *flag*

flag = *formobject*.**ShowAsDialog**

Legal values

Value	Description
FALSE	(Default) Do not display the form as a dialog box.
TRUE	Display the form as a dialog box.

Usage

Display a form as a dialog box when you want users to enter information, or when you want to get information from users before continuing.

For example, when users click the See Schedule for button in the Meeting Room Scheduler SmartMaster application, Approach displays a dialog box that was created as a form and then converted to a dialog box.

Approach: ShowInPreview property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SHOWINPREVIEW_EXSCRIPT',1)} See example

Sets or returns if a non-printing display element appears when previewing a view.

Data type

Integer

Syntax

displayelementobject.**ShowInPreview** = *flag*

flag = *displayelementobject*.**ShowInPreview**

Legal values

Value	Description
TRUE	(Default) Show the object while in Print Preview.
FALSE	Do not show the object while in Print Preview.

Usage

The ShowInPreview property is available only if the NonPrinting property for the display element is set to TRUE (the display element will not print).

Approach: Size property

{button ,AL(`H_LAS_FONT_CLASS;',0)} See list of classes

{button ,AL(`H_las_SIZE_EXSCRIPT',1)} See example

Sets or returns the size of the font type of the current display element in points.

Data type

Single

Syntax

fontobject.**Size** = *value*

value = *fontobject*.**Size**

or

displaylementobject.**Font.Size** = *value*

value = *displayelemetobject*.**Font.Size**

Legal values

The default font size is 10 points. Other legal values depend on the font.

Usage

Change the font size when you add a display element to make it consistent with other interface elements. For example, you can change the font size of a form title from 10 points to 14 points.

Approach: SlideLeft property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SLIDELEFT_EXSCRIPT',1)} See example

Sets or returns if a display element is arranged when printing so that it closes the gap with the display element to the left of it.

Data type

Integer

Syntax

displayelementobject.**SlideLeft** = *flag*

flag = *displayelementobject*.**SlideLeft**

Legal values

Value	Description
FALSE	(Default) Do not slide the display element to the left when printing.
TRUE	Slide the display element to the left when printing.

Usage

Slide a display element to the left if a display element to its left contains blank spaces that you do not want to appear on the printed page.

For example, a mailing label has a field City that is 30 characters long. If a user enters a name that is only 10 characters long, there are twenty blank spaces between the city and the state when the user prints the view. If you slide the State field to the left, the gap of 20 spaces is closed up.

Approach: SlideUp property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SLIDEUP_EXSCRIPT',1)} See example

Sets or returns if a display element is arranged when printing so that it closes the gap with the display element above it.

Data type

Integer

Syntax

displayelementobject.**SlideUp** = *flag*

flag = *displayelementobject*.**SlideUp**

Legal values

Value	Description
FALSE	(Default) Do not slide the display element up when printing.
TRUE	Slide the display element up when printing.

Usage

Slide a display element towards the top of the page if the display element above it contains blank spaces you do not want to appear on the printed page.

For example, you may create a field box object that is 200 characters long and underneath it, a picture object. If the user only enters 100 characters into the field box, there are 100 blank spaces between the text and the picture when the user prints the view. If you slide the picture object up, the gap of 100 spaces is closed up.

Approach: SQL property

{button ,AL(`H_LAS_QUERY_CLASS;',0)} See list of classes

{button ,AL(`H_las_SQL_EXSCRIPT',1)} See example

Sets or returns the SQL statement associated with the Query object.

Data type

String

Syntax

queryobject.**SQL** = *string*

string = *queryobject*.**SQL**

Legal values

Any valid SQL statement, formatted as a string.

Usage

Supply an SQL statement to run using the Execute method. In most cases, you supply an SQL SELECT statement to retrieve records from a table. Approach supports all SQL components of the SELECT statement. Some of the basic components of the SELECT statement are as follows:

Phrase	Description
SELECT *columnname* [,*columnname*]...	(Required) Return values from these columns.
FROM *tablename* [,*tablename*]...	(Required) Look for the columns in these tables.
WHERE (*findcondition* [,*findcondition*]...)	Return values that match these find conditions.

Approach requires the statement to be formatted as a string. If there are special characters in column or table names, you must enclose those names in double quotation marks. To have a double quotation mark inside the string, it must be preceded by another double quotation mark. For example, the following statement retrieves the Order Total column from an DB2 table called Orders:

```
Qry.SQL = "SELECT ""Order Total"" FROM Orders"
```

The first double quotation mark opens the string; the second and third produce a double quotation mark inside the string; the fourth and fifth also produce a double quotation mark in the string; and the sixth closes the string.

If the table name is stored in the variable MyTable, use a string concatenation character (&) to add the variable into the string:

```
Qry.SQL = "SELECT ""Order Total"" FROM " & MyTable
```

If the table name is lower case (on Oracle servers) or includes special characters, such as a full path designation, you must also include double quotation marks in the string to surround the table name. For example, if MyTable contained the string "orders!", the SQL statement looks like the following:

```
Qry.SQL = "SELECT ""Order Total"" FROM """ & MyTable & """"
```

The statement has three double quotation marks before the table name (the first two to generate a single double quotation mark and the third to close the string) and four double quotation marks after (the first to open the string, the second two to generate a single double quotation mark, and the fourth to close the string).

Adding find conditions to the statement in a WHERE clause requires careful attention to the placement of quotation marks. For example, the following statement returns Order records from after a date stored in the variable CutOffDate:

```
Qry.SQL = "SELECT ""Order Total"" FROM " & MyTable & _
   " WHERE (" & MyTable & "."Order Date"" > " & CutOffDate & ")"
```

The reference to the column Order Date is preceded by the table name and enclosed in double quotation marks.

If you indicate string values inside in the SQL string, enclose the string in single quotation marks.

This property is mutually exclusive of the TableName property. If an SQL statement is specified, Approach executes the SQL statement, clearing the TableName property setting.

Approach: Stretch property

{button ,AL(`H_LAS_PICTUREPLUS_CLASS;',0)} See list of classes

Sets or returns whether the picture in a PicturePlus field stretches if it is too small for the frame.

Data type

Integer

Syntax

pictureplusobject.**Stretch** = *flag*

flag = *pictureplusobject*.**Stretch**

Legal values

Value	Description
FALSE	(Default) The picture is not stretched to fit the PicturePlus field.
TRUE	The picture is stretched to fit the PicturePlus field

Usage

If the image is too small for the PicturePlus field, stretch it to fit.

Approach: Strikethrough property

{button ,AL(`H_LAS_FONT_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_STRIKETHRU_EXSCRIPT',1)} See example

Sets or returns the text attribute for the font of a display element to strikethrough.

Data type

Integer

Syntax

fontobject.**Strikethrough** = *flag*

flag = *fontobject*.**Strikethrough**

or

displaylementobject.**Font.Strikethrough** = *flag*

flag = *displayelemetobject*.**Font.Strikethrough**

Legal values

Value	Description
FALSE	(Default) Do not put a line through the font.
TRUE	Put a line through the font.

Approach: TabOrder property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABORDER_EXSCRIPT',1)} See example

Sets or returns the order in which display elements are selected when users press TAB to move around a form.

Data type

Integer

Syntax

displayelementobject.**TabOrder** = *value*

value = *displayelementobject*.**TabOrder**

Legal values

Set this property to any integer between 1 and the number of objects in the view.

The default tab order for most display elements is based on the order in which you add them to a view.

Display elements such as rectangles, text blocks, and so on, must be added to the tab order using this property.

Usage

Arrange the tab order so users can tab between elements in a logical order. For example, if you create a new field box, and add it to a form, it is last in the tab order regardless of where you placed it on the form.

Approach: TabStop property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABSTOP_EXSCRIPT',1)} See example

Sets or returns if a display element can be selected by users pressing TAB.

Data type

Integer

Syntax

displayelementobject.**TabStop** = *flag*

flag = *displayelementobject*.**TabStop**

Legal values

Value	Description
TRUE	(Default, except for ellipse, line, rectangle, rounded rectangle, and text blocks.) Allow tabbing to the object.
FALSE	Do not allow tabbing to the object.

Usage

Arrange display elements so they are or are not in the tab order.

For example, if you have a form with a picture field, and you do not want users to tab to the picture, turn off the TabStop property for that picture field.

Approach: Text property (Button class)

{button ,AL(`H_LAS_BUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_TEXT_BUTTON_EXSCRIPT',1)} See example

Sets or returns the text displayed on a button.

Data type

String

Syntax

buttonobject.**Text** = *stringexp*

stringexp = *buttonobject*.**Text**

Legal values

Any string up to 256 characters long.

Approach: Text property

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_TEXT_EXSCRIPT',1)} See example

Sets or returns the text value of the data in drop-down boxes, field boxes, list boxes, field boxes and lists, and text blocks.

Data type

String

Syntax

displayelementobject.**Text** = *string*

string = *displayelementobjet*.**Text**

Legal values

Any string up to 256 characters.

Usage

Determine the text in a display element so you can initiate another action.

For example, if users select CA in the State drop-down box, you can list only the Zip codes for California in the ZIP drop-down box.

Approach: Top property (Border class)

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

{button ,AL(`H_las_TOP_BORDER_EXSCRIPT',1)} See example

Sets or returns whether to display a border along the top of a display element.

Data type

Integer

Syntax

borderobject.**Top** = *flag*

flag = *borderobject*.**Top**

Legal values

Value	Description
TRUE	(Default) Display the top border.
FALSE	Do not display the top border.

Approach: Top property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_TOP_EXSCRIPT',1)} See example

Sets or returns the distance, measured in twips, between the top edge of a display element or panel and the top edge of its parent.

Data type

Long

Syntax

displayelementobject.**Top** = *value*

value = *displayelementobject*.**Top**

Legal Values

If the Top property is a negative number, the display element or panel may not appear in the view.

Approach: Type property

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHARTVIEW_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_REPORT_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS',0)} See list of classes

{button ,AL(`H_las_TYPE_EXSCRIPT',1)} See example

(Read-only) Returns the type of display element, panel, or view.

Data type

Long

Syntax

value = *object*.**Type**

Legal values

See the list of enumerators.

Usage

Determine the type of object.

For example, you may want to know if the current display element is a radio button or a check box.

Approach: UncheckedValue property

{button ,AL(`H_LAS_CHECKBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_UNCHECKEDVALUE_EXSCRIPT',1)} See example

Sets or returns the value associated with the unchecked state of a check box.

Note This value does not reflect the current state of the check box.

Data type

String

Syntax

*checkboxobject***.UncheckedValue** = *stringexp*

stringexp = *checkboxobject***.UncheckedValue**

Legal values

Any string up to 256 characters.

Usage

Determine the value of an unchecked check box, regardless of its current state. For example, you can display a set of radio buttons below only those check boxes that have an unchecked value of Don't know.

Approach: Underline property

{button ,AL(`H_LAS_FONT_CLASS;',0)} See list of classes

{button ,AL(`H_las_UNDERLINE_EXSCRIPT',1)} See example

Sets or returns the text attribute of the font of a display element to underline.

Data type

Integer

Syntax

fontobject.**Underline** = *flag*

flag = *fontobject*.**Underline**

or

displaylementobject.**Font.Underline** = *flag*

flag = *displayelemetobject*.**Font.Underline**

Legal values

Value	Description
FALSE	(Default) Do not underline the font.
TRUE	Underline the font.

Approach: UserID property

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_USERID_EXSCRIPT',1)} See example

Sets or returns the user name used to log on to a database server.

Data type

String

Syntax

connectionobject.**UserID** = *string*

string = *connectionobject*.**UserID**

Legal values

Any string.

Approach: Value property (CheckBox class)

{button ,AL(`H_LAS_CHECKBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_VALUE_CHECKBOX_EXSCRIPT',1)} See example

Sets or returns the value in the field represented by the check box, when you first define the checkbox.

Data type

String

Syntax

*checkboxobject***.Value** = *stringexp*

stringexp = *checkboxobject***.Value**

Legal values

Any value that is legal for the field, determined by its properties.

Usage

Set the initial value of the check box when you create it. You cannot change it's value once you have defined it.

Determine the value of the check box, and then perform another task based on the value.

For example, a view has five check boxes, and each has a value of 1 when checked and 0 when unchecked. You can add the total values together and run an appropriate macro based on the results.

Approach: Value property (RadioButton class)

{button ,AL(`H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

Sets or returns the value of the radio button based on whether it is turned on or off.

Data type

String

Syntax

radiobuttonobject.**Value** = *stringexp*

stringexp = *radiobuttonobject*.**Value**

Legal values

There is no default value for this property.

You can set this property to any value that is legal for the field, determined by its properties.

Usage

Set the on and off values of a new radio button, or when other data changes.

Determine the value of the radio button, then perform another task based on the value.

For example, a view has five radio buttons, and each has a value of one when turned on, and zero when turned off. You can add the total values together and run an appropriate macro based on the results.

Approach: Visible property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHARTVIEW_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPORT_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_VISIBLE_EXSCRIPT',1)} See example

Sets or returns whether a display element, panel, view, or application is visible.

Data type

Integer

Syntax

object.**Visible** = *flag*

flag = *object*.**Visible**

Legal values

Value	Description
TRUE	(Default) The object is visible.
FALSE	The object is not visible.

Approach: Width property (Border class)

{button ,AL(`H_LAS_BORDER_CLASS;',0)} See list of classes

{button ,AL(`H_las_WIDTH_BORDER_EXSCRIPT',1)} See example

Sets or returns the width of a solid line.

Data type

Long

Syntax

borderobject.**Width** = *value*

value = *borderobject*.**Width**

Legal values

Value	Description
$LtsBorderWidthNone	There is no border.
$LtsBorderWidthThin	The border is a thin line.
$LtsBorderThick	The border is a thick line.
AprHalfPoint	The border is 0.5 points wide.
Apr1Point	The border is 1 point wide.
Apr2Point	The border is 2 points wide.
Apr3Point	The border is 3 points wide.
Apr6Point	The border is 6 points wide.
Apr12Point	The border is 12 points wide.

Approach: Width property

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LINESTYLE_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_WIDTH_EXSCRIPT',1)} See example

Sets or returns the width, in twips, of a display element or panel.

Data type

Long

Syntax

displayelementobject.**Width** = *value*

value = *displayelementobject*.**Width**

or

panelobject.**Width** = *value*

value = *panelobject*.**Width**

Legal values

Maximum width: The maximum page size allowed by your page setup.

Approach: Yellow property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_YELLOW_EXSCRIPT',1)} See example

(Read-only) Returns the yellow component of a CMYK value for a display element.

Data type

Integer

Syntax

value = *colorobject*.**Yellow**

Legal values

Any integer from 0 to 255.

Usage

0 represents no yellow and 255 represents the highest amount of yellow available.

Approach: Activate method

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_ACTIVATE_EXSCRIPT',1)} See example

Makes the specified document (.APR file) active.

Syntax

Call *documentobject*.**Activate**()

Parameters

None

Return values

None

Usage

If you want users to access more than one document to complete a task, you can switch to the another document at the appropriate step in the procedure.

Approach: AddColumn method

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_ADDCOLUMN_EXSCRIPT',1)} See example

Adds a column to a worksheet.

Syntax

integer = *worksheetobject*.**AddColumn**(*fieldname, columnname*, *insertpos*)

Parameters

fieldname

A string representing the name of a field.

columnlabel

(Optional) A string representing the text for the column header.

insertpos

(Optional) A string representing the position to insert the column in the worksheet. The string is the header of the column to the left of the position for the new column.

Return values

Value	Description
TRUE	A new column was added to the worksheet.
FALSE	A new column was not added to the worksheet.

Approach: AddListItem method

{button ,AL(`H_LAS_LISTBOX_CLASS;',0)} See list of classes

Adds a new value to a list box.

Syntax

integer = *listboxobject*.**AddListItem**(*item*, [*index*])

Parameters

item

A string representing the value you want to add to the list of values displayed by the list box. If the string is longer than the length defined for the field associated with the list box, the string is truncated.

index

(Optional) An integer representing the location in the list that *item* is placed. If you do not specify an *index*, *item* is placed at the end of the list.

Return values

Value	Description
TRUE	The item was successfully added to the list.
FALSE	The item failed to be added to the list.

Usage

This method adds items to the list of choices available to the user.

Approach: AddRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_ADDROW_EXSCRIPT',1)} See example

Adds a new row (record) to a result set.

Syntax

Call *resultsetobject*.**AddRow**()

or

integer = *resultsetobject*.**AddRow**()

Parameters

None

Return values

Value	Description
TRUE	A new row was added to the result set.
FALSE	A new row was not added to the result set.

Usage

Add data to fields in the new row using the SetValue method.

The row is not committed to the result set until you call the UpdateRow method.

Approach: Add method (Collection class)

{button ,AL(`H_LAS_COLLECTION_CLASS;',0)} See list of classes

Adds an object to the current collection.

Syntax

index = *collectionobject*.**Add**(*object*)

Parameters

object

An object added to the collection.

Return values

Value	Description
Index	An integer representing the index for the new item added to the collection

Approach: Add method

{button ,AL(`H_LAS_FINDDISTINCT_CLASS;H_LAS_FINDDUPLICATE_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_ADD_EXSCRIPT',1)} See example

Adds a field to the find condition for a FindDistinct or FindDuplicate object.

Syntax

Call *findspecialobject*.**Add**(*field*)

Parameters

field

A string representing the name of the field.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

Return values

None

Usage

This method qualifies which fields are used to determine duplicate or distinct records. For example, create a FindDuplicate object using the FirstName field to find all records with the same first names. Then use the Add method to include the LastName field. The records found have duplicate first and last names.

Approach: Add method (Sort class)

{button ,AL(`H_LAS_SORT_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_ADD_SORT_EXSCRIPT',1)} See example

Adds another field to the sort condition for a Sort object.

Syntax

Call *sortobject*.**Add**(*field*, *order*)

Parameters

field

A string representing the name of the field.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

order

A long representing the order to sort the values in *field*. Choose from the following values:

Value	Description
LtsSortAscending	Sorts the field values from lowest to highest and from A to Z.
LtsSortDescending	Sorts the field values from highest to lowest and from Z to A.

Return values

None

Usage

This method adds fields to further refine the sort of the found set or the entire table. For example, create a Sort object using the LastName field as the primary field by which to sort the records in a table. Then use the Add method to include the FirstName field. The table is sorted by first and last name.

{button ,AL(`',1)} See example

Approach: And method

{button ,AL(`H_LAS_FIND_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_AND_EXSCRIPT',1)} See example

Adds a find condition to an existing Find object.

This new find condition is in an AND relationship with the previously added condition.

Syntax

Call *findobject*.**And**(*field*,*criteria*)

Parameters

field

A string representing the name of the field to use in this find condition.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

criteria

A string representing the find condition. For example, to search for records with values in *field* greater than 100, *criteria* has the value ">100".

Build criteria with wildcards, functions, operators, constants, and field references as you would other formulas in Approach.

Return values

None

Usage

This method adds a find condition to an existing Find object. The find conditions are in an AND relationship, which means that both conditions must be met by a record for that record to be included in the found set.

The following script shows a typical example of using the And method:

```
Sub FindState
   Set MyFind = New Find ("CONTACT", "Yoko Tanaka")
   Call MyFind.And ("STATE", "CA")
   CurrentDocument.Window.FindSort (MyFind)
End Sub
```

Approach: BringToFront method

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;H_LAS_DISPLAY_CLASS;',0)} See list of classes

{button ,AL(`H_las_BRINGTOFRONT_EXSCRIPT',1)} See example

Places a display element in front of all overlapping display elements.

Syntax

displayelementobject.**BringToFront**

Parameters

None

Return values

None

Usage

Arrange display elements so they do not interfere with entering data or hide other display elements.

For example, you can have a circle around a group of radio buttons. For users to be able to click the buttons, however, the buttons must be in front of the circle.

Approach: Browse method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Switches Approach to Browse.

Syntax

integer **=** *docwindowobject*.**Browse**

Parameters

None

Return values

Value	Description
TRUE	Approach successfully switched to Browse.
FALSE	Approach failed to switch to Browse.

Usage

Use this method to toggle the Approach environment between Print Preview and Browse.

Approach: Cascade method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;',0)} See list of classes

Cascades the windows. Using this LotusScript command is the same as choosing Window - Cascade.

Syntax

Call *applicationwindowobject*.**Cascade**()

Parameters

None

Return values

None

Approach: Close method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_WINDOW_CLASS;', 0)} See list of classes

{button ,AL(`H_las_CLOSE_EXSCRIPT',1)} See example

Closes the application or document window.

Using this LotusScript command is the same as clicking the Close button on the right of the application or document title bar.

Syntax

Call *applicationwindowobject*.**Close()**

Parameters

None

Return values

None

Usage

Use the Close method to close a form displayed as a dialog box.

For example, attach a script to a Cancel button to close a form used to enter dates.

Approach: Close method (ResultSet class)

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_CLOSE_RESULTSET_EXSCRIPT',1)} See example

Closes a result set.

Syntax

Call *resultsetobject*.**Close**()

or

integer = *resultsetobject*.**Close**

Parameters

None

Return values

Value	Description
TRUE	The result set was successfully closed.
FALSE	The result set was not closed.

Usage

To commit changes made to a result set, use the UpdateRow method.

Approach: ConnectTo method

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_CONNECTTO_EXSCRIPT',1)} See example

Connects to a data source.

Syntax

integer = *connectionobject*.**ConnectTo**(*source,* [*userid*], [*password*], *server*)

Parameters

source

A string identifying the file type to which you are connecting.

Choose from strings corresponding to the file types registered on your system. The strings are case-sensitive. Call ListDataSources method to determine the possible values.

The following values are some of the possible values:

dBASE III+
dBASE IV
FoxPro
Query
Lotus Notes - Local
Lotus Notes - Server
Lotus Notes - Workspace
Paradox
QMF Query
QMF Procedure
Oracle
SQL Server
ODBC Data Sources
Delimited Text
Fixed Length Text
Excel
1-2-3

userid

(Optional) A string representing a user ID. If the connection requires *userid* and you do not supply it, Approach prompts the user.

password

(Optional) A string representing a database server password corresponding to *userid*. If the connection requires *password* and you do not supply it, Approach prompts the user.

server

A string representing the server, data source name, or the path location for the data source.

Return values

Value	Description
TRUE	Approach successfully connected to the data source.
FALSE	Approach failed to connect to the data source.

Usage

The *userid* and *password* are saved for this data source during this script session so that subsequent connection

attempts to the same data source are automatically performed.

Use the New method to create a Connection object before calling ConnectTo.

This method allows you to specify connection information that Approach passes through to a client-server database. The requirements for each argument depend on the file type that you are connecting to. Some of these dependencies are as follows:

File type (*source*)	Data source name (*server*)	Behavior when connecting
ODBC Data Sources	(Required) A specific data source name, such as a specific SQL Server or DB2 data source. You must insert an exclamation point (!) before the data source name.	If more information is required to complete the connection or if the data source name could not be found, Approach prompts the user.
IBM DB2	(Required) A specific data source name.	If more information is required to complete the connection or if the data source name could not be found, Approach prompts the user.
SQL Server	(Required) A server name. This argument is required for the first connection on each client workstation. After the first connection, the server argument is required only if you change which server you are connecting to.	If a userid or password are required for the connection and you do not specify them, Approach prompts the user.
Oracle	(Required) A server name. This argument is required for the first connection on each client workstation. After the first connection, the server argument is required only if you change which server you are connecting to.	If a userid or password are required for the connection and you do not specify them, Approach prompts the user.
All others	(Optional) A path name. This argument sets the working directory for the connection for non-SQL file types. If you specify a path using this argument, you do not need to use a fully-qualified table name when defining a query associated with this connection.	If a userid or password are required for the connection and you do not specify them, Approach prompts the user.

For example, to connect to the DB2 Sample database, the ConnectTo statement looks like the following:

```
Result = Con.ConnectTo("ODBC Data Sources","UserID","Password","!Sample")
```

Or

```
Result = Con.ConnectTo("IBM DB2", "UserID", "Password", "Sample")
```

The exclamation point is only required when ODBC Data Sources is used as the *source* argument.

For Oracle7 servers, the ConnectTo statement looks like the following, where ORASRV is an Oracle7 server name:

```
Result = Con.ConnectTo("Oracle","UserID", "Password", "ORASRV")
```

For QMF Query or QMF Procedure connections, the ConnectTo statements look like the following:

```
Result = Con.ConnectTo("QMF Query")
Result = Con.ConnectTo("QMF Procedure")
```

Approach: CopyView method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Copies the specified views to the clipboard.

Syntax

integer = *docwindowobject*.**CopyView**(*views,* [*data*])

Parameters

views

Long representing the views to copy. Choose from the following constants:

Value	Description
aprCurrentView	Copy only the current view.
aprAllViews	Copy all views in the document.

data

(Optional) Long representing what data, if any, is copied with the view. Choose from the following constants:

Value	Description	Valid for
aprAllDatabases	Copy all data in the tables (database files) associate with the view.	All views
aprFoundSet	Copy only data in the current found set.	All views
aprCurrentRecord	Copy only the data displayed in the current record.	Form, Form Letter, Mailing Label, Envelope
aprEmptyTables	Do not copy any data.	All views

Return values

Value	Description
TRUE	The view is copied successfully.
FALSE	The view is not copied.

Usage

Use the CopyView method to paste an image of the view in another product, such as a word processor.

Approach: Copy method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Copies the selected text or picture to the Clipboard.

Syntax

Call *docwindowobject*.**Copy()**

Parameters

None

Return values

None

Usage

Use the Copy method with the Paste method.

CountRecords method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Returns the number of records in a found set or table.

Syntax

integer **=** *docwindowobject*.**CountRecords**([*datasettype*])

Parameters

datasettype

(Optional) Integer representing the set of records for which to find the number. Choose from the following constants:

Constant	Description
AprFoundRecords	(Default) Returns the number of records in the current found set.
AprAllRecords	Returns the number of records in the main table of the current view.

Return values

The number of records in the found set or table.

Usage

Use CountRecords when you are connecting to SQL tables, and the NumRecords or NumRecordsFound properties return -1.

Approach: CreateCalcField method

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

Create a calculated field associated with a document (.APR file).

Syntax

integer = *documentobject*.**CreateCalcField** (*fieldname*, *formula*)

Parameters

fieldname

A string representing the name of the new calculated field.

formula

A string representing the formula used to produce values for the calculated field. Valid formulas are the same as those you can create in the Field Definition dialog box.

Return values

Value	Description
TRUE	Approach successfully created the calculated field.
FALSE	Approach failed to create the calculated field.

Usage

Use a calculated field to calculate values based on field data, especially for calculating summaries or totals for grouped records.

Approach: CreateResultSet method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

Creates a ResultSet object from a table.

Syntax

Set *resultsetobject* = *tableobject*.**CreateResultSet**()

Parameters

None

Return values

Value	Description
ResultSet	A ResultSet object created from the table

Usage

This method lets you access data in a table already associated with an .APR file without using the Approach user interface. If the table is not already associated with an .APR file, use the Connection, Query, and ResultSet classes to produce a ResultSet object.

This method ignores the found set and produces a result set with all records from the table.

If the table was produced using the SQL Assistant, the ResultSet object reflects the subset opened in Approach through SQL, not the entire server table.

Approach: Cut Method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Copies the selected text or picture to the Clipboard and deletes the original text or picture.

Syntax

Call *docwindowobject*.**Cut()**

Parameters

None

Return values

None

Usage

Use the Cut method with the Paste method.

Approach: DeleteCalcField method

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

Deletes an existing calculated field in the document (.APR file).

Syntax

integer = *documentobject*.**DeleteCalcField** (*fieldname*)

Parameters

fieldname

A string representing the name of the calculated field to delete.

Return values

Value	Description
TRUE	Approach successfully deleted the specified calculated field.
FALSE	Approach failed to delete the specified field, probably because the field doesn't exist.

Usage

Use this method to clean up temporary or obsolete calculated fields.

This method fails if the specified field does not exist. If there are duplicate calculated field names, the first occurence of the filed name is deleted.

Approach: DeleteFile method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Deletes the specified file.

Syntax

integer = *docwindowobject*.**DeleteFile**(*filename* , [*nodialogs*])

Parameters

filename

A string representing the full path name of the file to be deleted.

nodialogs

(Optional) An integer indicating whether Approach should display error or confirmation dialog boxes to the user. Choose from the following values:

Value	Description
FALSE	(Default) Approach displays any error or confirmation dialog boxes to the user in response to deleting the file.
TRUE	Approach suppresses all error or confirmation dialog boxes that would display in response to deleting the file.

Return values

Value	Description
TRUE	Approach deleted the file successfully.
FALSE	Approach failed to delete the file.

Usage

Use this method to delete temporary files or files made obsolete by a script.

Note You cannot delete a file that is currently open.

Approach: DeleteFoundSet method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Deletes all records in the found set.

Syntax

Call *docwindowobject*.**DeleteFoundSet**([*noconfirmation*])

Parameters

noconfirmation

(Optional) An integer indicating whether or not Approach should display a confirmation dialog box to the user before deleting the records. Choose from the following values:

Value	Description
FALSE	(Default) Approach displays a confirmation dialog box to the user before deleting the records.
TRUE	Approach suppresses the display of a confirmation dialog box when deleting the records.

Return values

None

Usage

Use this method after creating a found set using a find request or by creating a Find, FindDistinct, or FindDuplicate object.

To delete all records in a table, create a new find using the Find method, use the asterisk wildcard as the criteria, then use the DeleteFoundSet method.

Approach: DeletePage method

{button ,AL(`H_LAS_FORM_CLASS;',0)} See list of classes

Removes a page from a form.

Syntax

integer = *formobject*.**DeletePage**(*pagenumber*)

Parameters

pagenumber

The page number to be deleted.

Return values

Value	Description
TRUE	The page was successfully deleted.
FALSE	The page was not successfully deleted.

Approach: DeleteRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Deletes the current record.

If the current view contains fields from more than one table, records are deleted from all tables according to the settings in the Relational Options dialog box, which were set when you joined the tables.

Syntax

Call *docwindowobject*.**DeleteRecord**([*noconfirmation*])

Parameters

noconfirmation

(Optional) An integer indicating whether to prompt the user to confirm that the record should be deleted. Choose from the following:

Value	Description
FALSE	(Default) Prompt the user to confirm the delete.
TRUE	Do not prompt the user for confirmation.

Approach: DeleteRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_DELETEROW_EXSCRIPT',1)} See example

Deletes the current row (record) in a result set.

Syntax

Call *resultsetobject*.**DeleteRow**()

integer = *resultsetobject*.**DeleteRow**()

Parameters

None

Return values

Value	Description
TRUE	The row in the result set was deleted.
FALSE	The row in the result set was not deleted.

Usage

This method fails under the following conditions:

- The result set is read-only.
- There is no valid result set to delete from.
- A result set is not cached.

Approach: Disconnect method

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_DISCONNECT_EXSCRIPT',1)} See example

Closes a connection to a specified data source type.

Syntax

Call *connectionobject*.**Disconnect**()

or

integer = *connectionobject*.**Disconnect**()

Parameters

None

Return values

Value	Description
TRUE	Approach successfully disconnected from the data source.
FALSE	Approach failed to disconnect from the data source.

Usage

Close the connection so other Connection objects can access the same data sources. If another connection is opened for the same data source, the second connection is read-only.

Approach: DoMenuCommand method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_DOMENUCOMMAND_EXSCRIPT',1)} See example

Executes an existing command from a menu.

Syntax

Call *applicationwindowobject*.**DoMenuCommand(***command*)

Parameters

command

An integer representing a menu command. The menu commands are defined as constants.

Return values

None

Usage

Execute any menu command available in the active application window. If a dialog box is associated with the menu command, it is displayed for the user to complete.

For example, if a custom menu is in operation, this method can only execute the commands available on the custom menu.

Approach: DoVerb method

{button ,AL(`H_LAS_OLEOBJECT_CLASS;',0)} See list of classes

Note DoVerb is not supported under OS/2.

Performs an OLE operation (verb) on an OLE object. The OLE server container determines the verbs available for the object.

Syntax

integer = *oleobject*.**DoVerb**(*verbnumber*)

Parameters

verbnumber

A number representing the verb to perform. Zero (0) is the primary verb, corresponding to the action performed when double-clicking the object. Click the object with the right mouse button to open a list of the available verbs. The primary verb (0) is listed first. Use *verbnumber* = 1 to perform the second verb in the list, and so on.

If *verbnumber* does not correspond to a verb available for the object, the primary verb is performed.

Return values

Value	Description
TRUE	The verb was performed.
FALSE	The verb was not performed.

Usage

Determine which OLE object verb to perform based on a specific action.

For example, you can perform an OLE object verb when users switch to a view. The primary verb for a wave sound OLE object is "Play." You can play the sound "Welcome" when users view the main menu in a document.

You can also perform an OLE object verb when users enter text in a field, select an item in a scrolling list, or click a button or graphic. The primary verb for a Freelance Graphics presentation is "Show." You can show a presentation when users click a "Demo" button.

Approach: DuplicateRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Creates a new record and copies the data from the current record into the new record.

Syntax

Call *docwindowobject*.**DuplicateRecord()**

Parameters

None

Return values

None

Usage

This method creates a new record containing all the data from the copied record. You can then edit the fields that require different data, or prompt the user for the new information.

Approach: Execute method

{button ,AL(`H_LAS_RESULTSET_CLASS;H_LAS_QUERY_CLASS;',0)} See list of classes

{button ,AL(`H_las_EXECUTE_EXSCRIPT',1)} See example

Executes a query.

Syntax

Call *object*.**Execute**()

or

integer = *object*.**Execute**()

Parameters

None

Return values

Value	Description
TRUE	The query or result set executed successfully.
FALSE	The query or result set failed to execute.

Usage

To create or refresh a result set, call the Execute method from a ResultSet object.

To run an SQL statement that does not produce a result set, call the Execute method from a Query object.

Retrieve information about a failed Execute method using the GetError method.

Approach: FieldExpectedDataType method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Sets or returns the data type in which a result set value is presented in Approach.

This "expected" data type can be different from the "native" data type in which the data is stored in the source table.

Syntax

integer = *resultsetobject*.**FieldExpectedDataType**(*column,* [*type*])

Parameters

column

A string representing the name of a column (field) in a result set.

Alternatively, you can use an integer representing the ID value of the column, if the name is unknown.

type

(Optional) An integer representing the expected data type.

Specify this parameter to set the expected data type. Leave it blank to return the currently set expected data type.

Return values

Value	Description
DB_BOOLEAN	The data type is a Boolean.
DB_CHAR	The data type is a string.
DB_DATE	The data type is a date.
DB_DOUBLE	The data type is a double.
DB_LONG	The data type is a long.
DB_SHORT	The data type is a short.
DB_TIME	The data type is a time.
DB_UNDEFINED	Resets the data type to the native data type.

Usage

The expected data type creates a conversion between the native data type for a column value and the data type expected by Approach.

Unless you specify an expected data type, Approach matches the native data type to the most appropriate equivalent when reading data from a table.

To determine the native data type for a column, use the FieldNativeDataType method.

Approach: FieldID method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the column ID for a column in a result set, given the column name (field name).

Syntax

integer = *resultsetobject*.**FieldID**(*columnname*)

Parameters

columnname

A string representing the name of a column in the result set.

Return values

Value	Description
Integer	The position of the specified column in the result set. For example, the first column in the result set has a column ID of 1.

Approach: FieldName method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_FIELDNAME_EXSCRIPT',1)} See example

Returns the field name associated with a column (field) in a result set, given the column ID.

Syntax

string = *resultsetobject*.**FieldName**(*columnid*)

Parameters

columnid

An integer representing the ID of a field. The first column in a result set has the column ID of 1, the second column has the value 2, and so on.

Return values

Value	Description
String	The name of the specified column.

Usage

The order of the fields in the result set is the same as their order on the source table unless you specify a new order in the SQL property of the query object that produced the result set.

Retrieve field names for an entire table using ListFields method from the Connection class.

Approach: FieldNativeDataType method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the native data type for the specified column (field) in a result set.

The native data type is the data type of the field as stored in the source table. The native data type can be different from the "expected" data type, which is the data type Approach uses to present the result set value.

Syntax

integer = *resultsetobject*.**FieldNativeDataType**(*column*)

Parameters

column

A string representing the name of a column (field) in a result set.

Alternatively, you can use an integer representing the ID value of the column, if the name is unknown.

Return values

Value	Description
SQL_BIT	The data type is a Boolean.
SQL_CHAR	The data type is a string.
SQL_DOUBLE	The data type is a double.
SQL_TIME	The data type is a time or date.
SQL_VARBINARY	The data type is a PicturePlus.
SQL_VARCHAR	The data type is a memo.

Approach: FieldSize method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the size of a column (field) from a result set.

Syntax

long = *resultsetobject*.**FieldSize**(*column*)

Parameters

column

A string representing the name of a column (field) in a result set.

Alternatively, you can use an integer representing the ID value of the column, if the name is unknown.

Return values

Value	Description
Long	The maximum number of characters allowed in the column.

Usage

This method returns sizes for text fields only. All other field types return -1.

Approach: FillField method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_FILLFIELD_EXSCRIPT',1)} See example

Fills a field in each record of the found set with the value you specify.

Using this LotusScript method is the same as executing the Fill Field command from the context menu.

Syntax

integer = *docwindowobject*.**FillField**(*fieldname, fillvalue*)

Parameters

fieldname

A string representing the specified field.

If there is more than one table (database file) joined in the document (.APR file), precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

"Products.Cost"

value

A string representing the value you want to use to update the field.

Return values

Value	Description
TRUE	The specified field was filled with the specified value.
FALSE	The specified field was not filled with the specified value.

Approach: FindAll method

{button ,AL(`H_LAS_docwindow_class;',0)} See list of classes

Makes available all of the records in all of the tables associated with a document (.APR file).

Syntax

Call *docwindowobject*.**FindAll()**

Parameters

None

Return values

None

Usage

After you do a find and you finish working with the found set, this method makes all the records in the database available to you.

Approach: FindSort method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_FINDSORT_EXSCRIPT',1)} See example

Executes the specified find or sort for the records in the DocWindow object.

Syntax

Call *docwindowobject*.**FindSort** (*findorsort* [,*sort*])

Parameters

findorsort

An existing Find, FindDistinct, FindDuplicate, FindTopLowest, or Sort object.

sort

(Optional) An existing Sort object.

Return values

None

Usage

After you define a Find, FindDistinct, FindDuplicate, FindTopLowest, or Sort object, use this method to execute the find or sort. Because the find or sort is defined separately from a document, you can use the FindSort method to associate any find or sort with any document, assuming that the find or sort makes sense for the document.

For example, use the New method for the FindDuplicate object to find records that have duplicate first, last, and company names. Then call the FindSort method for any document window in which you have tables that contain fields for first, last, and company names. The field names must be the same as those specified in the FindDuplicate object.

Approach: FirstRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_FIRSTRECORD_EXSCRIPT',1)} See example

Makes the first record in the found set the current record.

Using this LotusScript method is the same as clicking the First Record icon.

Syntax

Call *docwindowobject*.**FirstRecord**()

Parameters

None

Return values

None

Approach: FirstRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_FIRSTROW_EXSCRIPT',1)} See example

Sets the first row (record) in a result set as the current row.

Syntax

Call *resultsetobject*.**FirstRow**()

or

integer = *resultsetobject*.**FirstRow**()

Parameters

None

Return values

Value	Description
TRUE	The first row in the result set was set to be the current row.
FALSE	The current row was not changed.

Approach: GetAt method (FindTopLowest class)

{button ,AL(`H_LAS_FINDTOPLOWEST_CLASS;',0)} See list of classes

Retrieves the find condition for a top or lowest value find.

Syntax

integer = *findtoplowestobject*.**GetAt**(*field*, *countorpercent,* [*findtype*])

Parameters

field

A string into which this method writes the field used in the find condition.

countorpercent

An integer indicating the number of records to be returned. The number is a count or a percent based on the value of *findtype*.

findtype

(Optional) An integer representing the type of find performed. One of the following values is returned:

Value	Description
AprFindTop	(Default) The find returns the specified number of records representing the top values in the field.
AprFindTopPercent	The find returns the specified percent of records representing the top values in the field.
AprFindLowest	The find returns the specified number of records representing the lowest values in the field.
AprFindLowestPercent	The find returns the specified percent of records representing the lowest values in the field.

Return values

Value	Description
TRUE	The condition is retrieved successfully.
FALSE	The condition cannot be retrieved.

Approach: GetAt method (Find class)

{button ,AL(`H_LAS_FIND_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_GETAT_EXSCRIPT',1)} See example

Retrieves a find condition.

Syntax

integer = *findobject*.**GetAt**(*index*, *field*, *criteria*, [*ORNumber*])

Parameters

index

An integer indicating the find condition to be retrieved. The first condition has an *index* of zero.

field

A string into which this method writes the field used in the find condition.

criteria

A string into which this method writes the description of the find condition.

ORNumber

(Optional) An integer that this method returns to indicate how this find condition relates to the other find conditions.

If there is only one find condition, *ORNumber* is zero. As other conditions are added to the find, this number is incremented, depending on whether the conditions are related to the previous conditions in an AND or OR relationship.

If 2 conditions have	Then they are in an	And to be part of the found set
The same ORNumber	AND relationship	A record must satisfy both conditions.
Different ORNumbers	OR relationship	A record must satisfy at least one condition.

Return values

Value	Description
TRUE	The find condition is retrieved successfully.
FALSE	The find condition cannot be retrieved.

Usage

This method displays the conditions of a Find object. To show all of the find conditions, call GetAt in a loop. Use the result from GetCount as the upper bound of the loop.

If you have a FindDistinct, FindDuplicate, or FindTopLowest object associated with the Find object through the FindSpecial property, you must use the GetAt method for the FindDistinct, FindDuplicate, or FindTopLowest object to list those find conditions.

Approach: GetAt method

{button ,AL(`H_LAS_FINDDISTINCT_CLASS;H_LAS_FINDDUPLICATE_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_GETAT_EXSCRIPT',1)} See example

Retrieves a find condition.

Syntax

integer = *findspecialobject*.**GetAt**(*index*, *field*)

Parameters

index

An integer indicating the find condition to be retrieved. The first condition has an *index* of zero.

field

A string into which this method writes the field used in the find condition.

Return values

Value	Description
TRUE	The find condition is retrieved successfully.
FALSE	The find condition cannot be retrieved.

Usage

This method displays the conditions of a FindDistinct or FindDuplicate object. To show all of the find conditions, call GetAt in a loop. Use the result from GetCount as the upper bound of the loop.

For information about retrieving conditions for Find, Sort, or FindTopLowest objects, see the GetAt (Find class), GetAt (Sort class), and GetAt (FindTopLowest) methods.

Approach: GetAt method (Sort class)

{button ,AL(`H_LAS_SORT_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_GETAT_EXSCRIPT',1)} See example

Retrieves a sort condition.

Syntax

integer = *sortobject*.**GetAt**(*index*, *field*, *order*)

Parameters

index

An integer indicating the sort condition to be retrieved. The first condition has an *index* of zero.

field

A string into which this method writes the field used in the sort condition.

order

A long representing the order in which the field values are sorted. The following values are returned:

Value	Description
LtsSortAscending	Sorts the field values from lowest to highest and from A to Z.
LtsSortDescending	Sorts the field values from highest to lowest and from Z to A.

Return values

Value	Description
TRUE	The sort condition is retrieved successfully.
FALSE	The sort condition cannot be retrieved.

Usage

This method displays the conditions of a Sort object. To show all of the conditions, call GetAt in a loop. Use the result from GetCount as the upper bound of the loop.

Approach: GetColorFromRGB method

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETCOLORFROMRGB_EXSCRIPT',1)} See example

Assigns a color when given an RGB value.

Many of the RGB values are defined as constants.

Syntax

Set *colorobject* = *applicationobject*.**GetColorFromRGB**(*rgbvalue*)

Parameters

rgbvalue

A long representing an RGB value.

Return values

Value	Description
Color	Returns a color object.

Usage

Change the color of a display element by assigning it one of the application RGB colors.

For example, you can change the background color of a field depending on the importance of the information, such as optional as opposed to required fields.

Approach: GetCount method

{button ,AL(`H_LAS_FIND_CLASS;H_LAS_FINDDISTINCT_CLASS;H_LAS_FINDDUPLICATE_CLASS;H_LAS_SORT_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_GETCOUNT_EXSCRIPT',1)} See example

Retrieves the number of find conditions.

Syntax

integer = *findobject*.**GetCount**()

Parameters

None

Return values

An integer representing the number of find or sort conditions described by the specified Find, FindDistinct, FindDuplicate, or Sort object.

Usage

A find consists of one or more find conditions, each corresponding to a field. The GetCount method retrieves the total number of conditions in a given Find, FindDistinct, FindDuplicate, or Sort object. The GetAt method for each of these classes retrieves the description of each condition.

Approach: GetErrorMessage method

{button ,AL(`H_LAS_CONNECTION_CLASS;H_LAS_QUERY_CLASS;H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns a short text message associated with an error code.

Syntax

string = *object*.**GetErrorMessage**([*errorvalue*])

Parameters

errorvalue

(Optional) An integer representing the error code for which you want to get the text message.

If *errorvalue* is not specified, Approach returns the error message for the last error encountered.

The GetError method returns the most recent *errorvalue* for an object. The possible error codes are listed with the Approach predefined constants.

Return values

Value	Description
String	The text for the specified error code value

Usage

This method allows you to pass error messages to users. For example, if you are creating a ResultSet object from user input, use the result of this method to prompt the user if the input fails to create a valid ResultSet object.

Approach: GetError method

{button ,AL(`H_LAS_CONNECTION_CLASS;H_LAS_QUERY_CLASS;H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the most recent error code, which indicates the reason for the failure to execute a Connection, Query, or ResultSet object.

Syntax

integer = *object*.**GetError**()

Parameters

None

Return values

Any of the error codes listed with the Approach predefined constants.

Usage

Use the return value from this method as an input to the GetErrorMessage or GetExtendedErrorMessage methods.

Approach: GetExtendedErrorMessage method

{button ,AL(`H_LAS_CONNECTION_CLASS;H_LAS_QUERY_CLASS;H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns a long text message associated with an error code.

Syntax

string = *object*.**GetExtendedErrorMessage**([*errorvalue*])

Parameters

errorvalue

(Optional) An integer representing the error code for which you want to get the text message.

If *errorvalue* is not specified, Approach returns the error message for the last error encountered.

The GetError method returns the most recent *errorvalue* for an object. The possible error codes are listed with the Approach predefined constants.

Return values

Value	Description
String	The text for the specified error code value.

Usage

This method passes error messages to users. For example, if you are creating a ResultSet object from user input, use this method to prompt the user if the input fails to create a valid ResultSet object.

Approach: GetFieldFormula method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETFIELDFORMULA_EXSCRIPT',1)} See example

Returns the formula definition for a calculated field.

Syntax

string = *calctableobject*.**GetFieldFormula**(*fieldname*)

Parameters

fieldname

A string representing the name of a calculated field. The *fieldname* is not case-sensitive.

Return values

Value	Description
String	The formula defined for the field in the Field Definition dialog box.

Usage

This method returns the contents of a calculated field table identified by the CalcTable property of the Document class.

Approach: GetFieldOptions method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETFIELDOPTIONS_EXSCRIPT',1)} See example

Returns the options used to define a field, for example, the default value and validation criteria.

Syntax

string = *tableobject*.**GetFieldOptions**(*fieldname*)

Parameters

fieldname

A string representing a field in the table.

Return values

Value	Description
String	The options specified in the Formula/Options column for the field in Field Definition dialog box

Approach: GetFieldSize method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETFIELDSIZE_EXSCRIPT',1)} See example

Returns the size of a field.

Syntax

integer = *tableobject*.**GetFieldSize**(*fieldname*)

Parameters

fieldname

A string representing a field in the table.

Return values

Value	Description
Integer	The number of characters or digits that the field can hold

Usage

This method returns a valid size for text fields only. All other data types return a value of -1.

Approach: GetFieldType method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETFIELDTYPE_EXSCRIPT',1)} See example

Returns the type of a field.

Syntax

long = *tableobject*.**GetFieldType**(*fieldname*)

Parameters

fieldname

A string representing a field in the table.

Return values

Value	Description
AprFieldBool	Boolean field
AprFieldCalculation	Calculated field
AprFieldDate	Date field
AprFieldMemo	Memo field
AprFieldNumber	Numeric field
AprFieldPicture	PicturePlus field
AprFieldText	Text field
AprFieldTime	Time field
AprFieldVariable	Variable field

Approach: GetFuriganaField method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

This method is not used in this version of Approach.

Approach: GetFuriganaMode method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

This method is not used in this version of Approach.

Approach: GetHandle method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_WINDOW_CLASS;', 0)} See list of classes

{button ,AL(`H_las_GETHANDLE_EXSCRIPT',1)} See example

Returns a window handle to the window.

Syntax

long = *windowobject*.**GetHandle**

Parameters

None

Return values

Value	Description
Long	Returns a window handle.

Usage

For more information on using this method, see your operating system application development reference documentation.

Approach: GetIMEMode method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

This method is not used in this version of Approach.

Approach: GetIMEState method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

This method is not used in this version of Approach.

Approach: GetParameterName method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the name of an SQL parameter, given the parameter index.

Syntax

string = *resultsetobject*.**GetParameterName**(*parameterindex*)

Parameters

parameterindex

An integer representing the ordinal position of the parameter. The first parameter in the SQL statement has an index of 1, the second has an index of 2, and so on.

Return values

Value	Description
String	The name of the specified parameter, without the enclosing question marks (?).

Usage

In SQL, a parameter is a variable that is used in an SQL statement. The parameter name appears in the SQL statement surrounded by question marks (?). Use this method to return the parameter name.

The parameter names can be used to prompt a user for missing SQL information.

Approach: GetParameter method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the last value set for a specified SQL parameter.

Syntax

string = *resultsetobject*.**GetParameter**(*parameter*)

Parameters

parameter

A string representing the name of the parameter.

Alternatively, you can use an integer representing the ordinal position of the parameter, if the name is unknown. The first parameter in the SQL statement has an index of 1, the second has an index of 2, and so on.

Return values

Value	Description
String	The value of the parameter

Usage

In SQL, a parameter is a variable that is used in an SQL statement. The parameter name appears in the SQL statement surrounded by question marks (?). Use this method to return the current value of the parameter.

Change the value of a parameter using the SetParameter method.

Unlike ODBC and standard SQL, parameters used in Approach queries can appear anywhere in the SQL statement.

If the specified parameter name or index is invalid, Approach displays an error message and no value is returned.

Approach: GetRGB method

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

Returns the RGB value of the color.

Syntax

long = *colorobject*.**GetRGB**()

Parameters

None

Return values

Value	Description
Long	Returns the red, green and blue values of the object color object.

Usage

Determine the RGB value of a display element so you can set another display element to the same color using the SetRGB method.

Approach: GetTableByName method

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETTABLEBYNAME_EXSCRIPT',1)} See example

(Read only) Returns the details of the current table in the document (.APR file).

Syntax

Set *tableobject* = *documentobject*.**GetTableByName**(*tablename*)

Parameters

tablename

A string representing the name of the selected table.

Return values

Value	Description
Table	Returns the selected table.

Usage

Determine details of the table, such as its file name, path, number of fields and records, and field names.

Approach: GetText method

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETTEXT_EXSCRIPT',1)} See example

Returns the text in the current row for the specified column. If there is no current row, then an empty string is returned.

Syntax

string = *worksheetobject*.**GetText**([*column*])

Parameters

column

(Optional) A string representing the header of the column. The header, or label, may differ from the column's field name.

Return values

Value	Description
Data	A string representing the text returned from the current row.

Approach: GetValue method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_GETVALUE_EXSCRIPT',1)} See example

Returns the value in the current row (record) of the specified column (field) of the result set.

Syntax

variant = *resultsetobject*.**GetValue**(*column*)

Parameters

column

A string representing the name of a column (field) in a result set.

Alternatively, you can use an integer representing the ID value of the column, if the name is unknown.

Return values

Value	Description
Variant	The value in the specified column of the current row

Usage

The current row of the result set and the specified column identify a cell in the source table.

The data type of the returned value is determined by the expected data type set for this column by the FieldExpectedDataType method. For example, the expected value can be set to return a real value for an integer numeric field.

Approach: GoToRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Makes the specified record the current record.

Syntax

integer = *docwindowobject*.**GoToRecord**(*recordnumber*)

Parameters

recordnumber

A long representing the record number to display.

Return values

Value	Description
TRUE	Approach successfully changed to the record specified.
FALSE	Approach failed to change to the record specified, probably because that record does not exist.

Usage

This method automates clicking the record number in the status bar to display the Go to Record dialog box.

Approach: HideRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Hides the current record.

Syntax

Call *docwindowobject*.**HideRecord()**

Parameters

None

Return values

None

Usage

This method temporarily removes the current record from the current found set, sorts, or calculations. The user cannot print or delete a hidden record. Restore hidden records by doing a find, including a find that returns all records.

Approach: InsertAfter method

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_INSERTAFTER_EXSCRIPT',1)} See example

Places a display element directly behind one you specify as the parameter.

Syntax

integer = *displayelementobject*.**InsertAfter**(*object*)

Parameters

object

A display element object to be in front.

Return values

Value	Description
FALSE	(Default) The display element is not inserted after another.
TRUE	The display element is inserted after another.

Usage

Arrange display elements so they do not hide other display elements.

For example, there is a large rectangle with a circle in front of it. You just created a medium sized square and placed it over the circle. You want to place the square behind the circle, but in front of the rectangle, so you would insert the square after the circle.

Approach: IsCommandChecked method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_ISCOMMANDCHECKED_EXSCRIPT',1)} See example

Returns whether a menu command has a check mark next to it.

Syntax

integer = *applicationwindowobject*.**IsCommandChecked**(*command*)

Parameters

command

An integer representing the command you want to evaluate. The commands that can be checked are defined as constants.

Return values

Value	Description
TRUE	The menu command is checked.
FALSE	The menu command is not checked.

Usage

Use this method for any of the menu commands that users can check in the user interface.

For example, you can check to see if a user has "Show View Tabs" checked.

Approach: IsCommandEnabled method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_ISCOMMANDENABLED_EXSCRIPT',1)} See example

Returns whether a menu command is available (not dimmed).

Syntax

integer = *applicationwindowobject*.**IsCommandEnabled**(*command*)

Parameters

command

An integer representing the menu command you want to evaluate. The menu commands that can be enabled are defined as constants.

Return values

Value	Description
TRUE	The menu command is available.
FALSE	The menu command is not available.

Usage

Determine if a menu command is available so you can execute the menu command. For example, you cannot delete a found set if users have not created one, so the menu command is not available.

Approach: IsEmpty method

{button ,AL(`H_LAS_BASECOLLECTION_CLASS;H_LAS_COLLECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_ISEMPTY_EXSCRIPT',1)} See example

Determines if a BaseCollection object or Collection object is empty.

Syntax

integer = *basecollectionobject*.**IsEmpty**()

or

integer = *collectionobject*.**IsEmpty**()

Parameters

None

Return values

Value	Description
TRUE	The collection was empty.
FALSE	The collection was not empty.

Approach: LastRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_LASTRECORD_EXSCRIPT',1)} See example

Makes the last record in the found set the current record.

Using this LotusScript method is the same as clicking the Last Record icon.

Syntax

Call *docwindowobject*.**LastRecord**()

Parameters

None

Return values

None

Approach: LastRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_LASTROW_EXSCRIPT',1)} See example

Sets the last row (record) to be the current row in the result set.

Syntax

Call *resultsetobject*.**LastRow**()

or

integer = *resultsetobject*.**LastRow**()

Parameters

None

Return values

Value	Description
TRUE	The last row was set to be the current row.
FALSE	The current row did not change.

Approach: ListDataSources method

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_LISTDATASOURCES_EXSCRIPT',1)} See example

Returns a list of all registered data source types.

Syntax

variant = *connectionobject*.**ListDataSources**()

Parameters

None

Return values

Value	Description
Variant	An array of strings listing the registered data source types.

Usage

This method indicates the exact syntax required for the *source* parameter of the ConnectTo method.

Approach: ListFields method

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_LISTFIELDS_EXSCRIPT',1)} See example

Returns a list of the field names in the specified table.

Syntax

variant = *connectionobject*.**ListFields**(*source*)

Parameters

source

A string representing the name of a table accessible through the Connection object.

Return values

Value	Description
Variant	An array of strings listing the fields in a table

Usage

For non-SQL tables, you must use the complete path if the table is not in Approach's working directory. For example:

```
MyList = Con.ListFields("c:\projdata\testdb.dbf")
```

Approach: ListTables method

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_LISTTABLES_EXSCRIPT',1)} See example

Returns the names of the available tables of a specified data source type.

Syntax

variant = *connectionobject*.**ListTables**([*category*])

Parameters

category

(Optional) A string representing the data source type.

For SQL data source types, use *category* to specify a data source type. The ListDataSources method returns appropriate values for *category*. If you do not specify *category*, the method uses the data source type associated with the Connection object.

For non-SQL data source types, use *category* to specify a directory from which to list the available tables. If you do not specify *category*, the method returns a list of the tables from the current working directory.

Return values

Value	Description
Variant	An array of strings representing the table names

Approach: MakeNamedStyle method

{button ,AL(`H_LAS_BODYPANEL_CLASS;H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_HEADERFOOTERPANEL_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PANEL_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_SUMMARYPANEL_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_MAKENAMEDSTYLE_EXSCRIPT',1)} See example

Creates a named style from the display element's attributes.

Syntax

integer = *object*.**MakeNamedStyle**(*name*)

Parameters

name

A string representing the name of the named style that you create.

Return values

Value	Description
TRUE	The named style is created.
FALSE	The named style is not created.

Usage

Create a named style from the attributes of the current display element so that you can apply that element's attributes to other display elements.

For example, if a text box has red, bold text, create a named style named RedBold.

Approach: Maximize method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_WINDOW_CLASS;', 0)} See list of classes

{button ,AL(`H_las_MAXIMIZE_EXSCRIPT',1)} See example

Maximizes the application or document window.

Using this LotusScript method is the same as clicking the Maximize button.

Syntax

Call *applicationwindowobject*.**Maximize**()

or

Call *docwindowobject*.**Maximize**()

Parameters

None

Return values

None

Approach: Merge method

{button ,AL(`H_LAS_COLLECTION_CLASS;',0)} See list of classes

Combines the list of objects from another collection with the current collection.

Syntax

integer = *collectionobject*.**Merge**(*othercollection*)

Parameters

othercollection

A base collection representing each element in the collection.

Return values

Value	Description
integer	The total number of items in the collection

Usage

Use this method when you want to combine all the objects from two collections into one collection. The method returns the size (count) of the updated collection. No attempt is made to ensure uniqueness. If both collections contain the same object, it appears in the merged collection twice.

For example, if you have two forms, you can merge the collection of objects from each form to create one form. Each object in the specified collection is added to the collection of the current form.

Approach: Minimize method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_WINDOW_CLASS;', 0)} See list of classes

{button ,AL(`H_las_MINIMIZE_EXSCRIPT',1)} See example

Minimizes the application or document window.

Using this LotusScript method is the same as clicking the Minimize button.

Syntax

Call *applicationwindowobject*.**Minimize**()

or

Call *docwindowobject*.**Minimize**()

Parameters

None

Return values

None

Approach: NewPage method

{button ,AL(`H_LAS_FORM_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEWPAGE_EXSCRIPT',1)} See example

Adds a new page to a form.

Syntax

formobject **.NewPage**()

value = *formobject***.NewPage**()

Parameters

None

Return values

Value	Description
TRUE	A new page was added to the form.
FALSE	A new page was not added to the form.

Usage

This method adds new pages to forms. You can add up to 5 pages.

Approach: NewRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEWRECORD_EXSCRIPT',1)} See example

Creates a new record.

Using this LotusScript method is the same as clicking the New Record icon.

Syntax

Call *docwindowobject*.**NewRecord**()

Parameters

None

Return values

None

Approach: New method (Button class)

{button ,AL(`H_LAS_BUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_BUTTON_EXSCRIPT',1)} See example

Creates a new button.

Syntax

Set *buttonobject* = **New Button** (*parent*, [*text*], [*pagenum*])

Parameters

parent

An object from the Panel class that specifies which panel is the button's parent, and so identifies where the button displays.

For example, the button can be placed in a report header or footer, or in the body panel.

text

(Optional) The text you want to appear on the button.

pagenum

(Optional) An integer representing the page number of the view on which to place the button.

Return values

The new object is called ObjButton. A number is added to the name if a button with that name already exists on the same panel.

For example, the name of the first button placed on the body of a form appears as follows:

```
ObjButton
```

The name of the second button appears as follows:

```
ObjButton1
```

Approach: New method (ChartView class)

{button ,AL(`H_LAS_CHARTVIEW_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_CHARTVIEW_EXSCRIPT',1)} See example

Creates a new ChartView object.

Syntax

Set *chartviewobject* = **New ChartView**(*parent, xaxis, yaxis, series, calculation,* [*charttype*], [*maintable*]*)*

Parameters

parent

The Document object that contains the chart view.

xaxis

A string array representing the field used to plot the x-axis of the chart.

yaxis

A string array representing the field or fields used to plot the y-axis of the chart.

series

A string array representing the field or fields used to determine the chart legend.

calculation

A long representing the type of calculation. The calculation types are provided as enumerators.

charttype

(Optional) A long representing the chart type. The chart types are provided as enumerators.
If you don't specify *charttype*, Approach creates a bar chart.

maintable

(Optional) A string representing the name of the main table on which the chart is based.

Return values

Value	Description
ChartView	Returns a new chart

Usage

Create a chart when you want to generate the chart dynamically rather than storing pre-made chart variations.

Approach: New method (CheckBox class)

{button ,AL(`H_LAS_CHECKBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_CHECKBOX_EXSCRIPT',1)} See example

Creates a new check box.

Use CheckBox object properties to position the check box, create and style labels, and set the checked and unchecked values of the check box.

Syntax

Set *checkboxname* = **New CheckBox** (*parent,* [*pagenum*])

Parameters

parent

The panel that contains the check box. A check box can be placed on the body of a form or in the footer, header, summary, or body panel of a report.

Use dot notation to refer to the Panel object.

pagenum

(Optional) For forms only. An integer (1-5) representing the page that contains the check box.

Return values

The new object is called ObjCheckBox. A number is added to the name if a check box with that name already exists on the same panel.

For example, the name of the first check box placed on the body of a form appears as follows:

```
ObjCheckBox
```

The name of the second check box appears as follows:

```
ObjCheckBox1
```

Approach: New method (Collection class)

{button ,AL(`H_LAS_COLLECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_COLLECTION_EXSCRIPT',1)} See example

Creates a new collection.

Syntax

Set *collectionobject* = **New Collection**()

Parameters

None

Return values

Value	Description
Collection	Returns a new collection

Approach: New method (Color class)

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_COLOR_EXSCRIPT',1)} See example

Creates a new Color object with the red, green, and blue values that you specify.

Syntax

Set *color* = **New Color**([*red*], [*green*], [*blue*], [*transparent*])

Parameters

red

(Optional) An integer representing the amount of red present in the new color. Values range from 0 to 255. The default value is 0, representing no red.

green

(Optional) An integer representing the amount of green present in the new color. Values range from 0 to 255. The default value is 0, representing no green.

blue

(Optional) An integer representing the amount of blue present in the new color. Values range from 0 to 255. The default value is 0, representing no blue.

transparent

(Optional) An integer representing the amount of transparent color. Values range from 0 to 255. The default value is 255, representing transparent color.

Return values

Value	Description
Color	A new color object with the specified red, green, and blue values.

Usage

If no values are specified, the values for red, blue, and green are set to 0 so the Color object is black.

Approach: New method (Connection class)

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

Creates a new instance of the Connection class.

Syntax

Set *connectionobject* = **New Connection**()

Parameters

None

Return values

Value	Description
Connection object	A new instance of the Connection class.

Usage

After creating a Connection object, you can complete the following operations:

- Opening the connection using the ConnectTo method
- Determining the available data source types using ListDataSources method
- Setting commit behavior using the AutoCommit property and Transactions method

Approach: New method (Crosstab class)

{button ,AL(`H_LAS_CROSSTAB_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_CROSSTAB_EXSCRIPT',1)} See example

Creates a new Crosstab object with the rows, columns, labels, and calculations that you specify.

Syntax

Set *crosstabobject* = **New Crosstab**(*parent, rowfields, colfields, bodyfields, bodycalc, rowtotallabel, rowtotalcalc, coltotallabel, coltotalcalc,* [*maintable*])

Parameters

parent

The document (.APR file) that contains the crosstab.

rowfields

A string array representing the row field(s) used to create the rows of the crosstab.

colfields

A string array representing the field(s) used to create columns of the crosstab.

bodyfields

A string array representing fields to be calculated in the body of the crosstab.

bodycalc

A long representing the calculation type for the body of the crosstab. Calculation types are provided as enumerators.

rowtotallabel

A string representing the label for a summary row.

rowtotalcalc

A long representing the calculation type for a summary row. Calculation types are provided as enumerators.

coltotallabel

A string representing the label for a summary column.

coltotalcalc

A long representing the calculation type for a summary column. Calculation types are provided as enumerators.

maintable

(Optional) The main table for the crosstab.

Return values

Value	Description
Crosstab	Returns a new crosstab.

Approach: New method (Document class)

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_DOCUMENT_EXSCRIPT',1)} See example

Creates a new document (.APR file) based on a ResultSet object. The ResultSet becomes the main table for the document.

Syntax

Set *documentobject* = **New Document**(*resultset*)

Parameters

resultset

An existing ResultSet object.

Return values

Value	Description
Document	A new Document object based on a ResultSet object.

Usage

Use this method to create an .APR file from the results of a query. Create a ResultSet object using the Connection, Query, and ResultSet classes.

Use Document object properties to add a description and enter keywords for the new document.

Approach: New method (DropDownBox class)

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_DROPDOWNBOX_EXSCRIPT',1)} See example

Creates a new drop-down box.

Syntax

Set *dropdownboxname* = **New DropDownBox**(*parent*, [*pagenum*])

Parameters

parent

A Panel object that contains the drop-down box. A drop-down box can be placed on a form or in the body panel of a report.

*page*num

(Optional) An integer representing the page that contains the drop-down box.

Return values

The new drop-down box returns the name of the field to which it is bound.

Approach: New method (Ellipse class)

{button ,AL(`H_LAS_ELLIPSE_CLASS;',0)} See list of classes

Creates a new ellipse.

Use Ellipse object properties to set the color of the ellipse border, background, and shadow, and determine whether or not the ellipse is in the tab order.

Syntax

Set *ellipsename* = **New Ellipse**(*panel,* [*pagenum*])

Parameters

panel

The panel that contains the ellipse. Place an ellipse on the body of a form, mailinig label, form letter, envelope, or chart; or in the footer, header, summary, or body panel of a report.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the ellipse.

Return values

The new object is called ObjEllipse. A number is added to the name if an ellipse with that name already exists on the same panel.

For example, the name of the first ellipse placed on the body of a form appears as follows:

ObjEllipse

The name of the second ellipse appears as follows:

ObjEllipse1

Approach: New method (FieldBox class)

{button ,AL(`H_LAS_FIELDBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_FIELDBOX_EXSCRIPT',1)} See example

Creates a new field box.

Syntax

Set *fieldboxobject* = **New FieldBox**(*panel*, [*pagenum*])

Parameters

panel

A Panel object that contains the field box. A field box can be placed on a form or in the body panel of a report.

pagenum

(Optional) An integer (1 - 5) representing the page that contains the field box.

Return values

The return value is the FieldBox object.

Approach: New method (FindDistinct class)

{button ,AL(`H_LAS_FINDDISTINCT_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_NEW_FINDDISTINCT_EXSCRIPT',1)} See example

Creates a FindDistinct object.

Syntax

Set *finddistinctobject* = **New FindDistinct** ([*field*])

Parameters

field

(Optional) A string representing the name of the field in which to find distinct values.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

Return values

Value	Description
FindDistinct	A new FindDistinct object

Usage

This method creates a FindDistinct object. When you execute the script containing the FindDistinct object, Approach creates a found set of records that have distinct values in the field specified.

You can add more fields to the find by calling the Add method.

Approach: New method (FindDuplicate class)

{button ,AL(`H_LAS_FINDDuplicate_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_NEW_FINDDUPLICATE_EXSCRIPT',1)} See example

Creates a FindDuplicate object.

Syntax

Set *findduplicateobject* = **New FindDuplicate** ([*field*])

Parameters

field

(Optional) A string representing the name of the field to find duplicate values in.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

Return values

Value	Description
FindDuplicate	A new FindDuplicate object

Usage

This method creates a FindDuplicate object. When you execute the script containing the FindDuplicate object, Approach creates a found set of records that have the same values in the field specified.

You can qualify the find by adding more fields by calling the Add method.

Approach: New method (FindTopLowest class)

{button ,AL(`H_LAS_FINDtoplowest_CLASS;',0)} See list of classes

Creates a FindTopLowest object.

Syntax

Set *findtoplowestobject* = **New FindTopLowest** (*field, countorpercent,* [*findtype*])

Parameters

field

A string representing the name of the field to find top or lowest values in.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

countorpercent

An integer indicating the number of records to be returned. The number is a count or a percent based on the value of *findtype*.

findtype

(Optional) An integer representing the type of find performed. If no *findtype* is specified, Approach performs a top-value find by count. Choose from the following values:

Value	Description
AprFindTop	(Default) The find returns the specified number of records representing the top values in the field.
AprFindTopPercent	The find returns the specified percent of records representing the top values in the field.
AprFindLowest	The find returns the specified number of records representing the lowest values in the field.
AprFindLowestPercent	The find returns the specified percent of records representing the lowest values in the field.

Return values

Value	Description
FindTopLowest	A new FindTopLowest object.

Usage

This method creates a FindTopLowest object. When you execute the script containing the FindTopLowest object, Approach creates a found set of records that have the top or lowest values in the field, determined by count or percent.

Approach: New method (Find class)

{button ,AL(`H_LAS_FIND_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_NEW_FIND_EXSCRIPT',1)} See example

Creates a Find object.

Syntax

Set *findobject* = **New Find** ([*field*], [*criteria*])

Parameters

field

(Optional) A string representing the name of the field to use in this find condition.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

criteria

(Optional) A string representing the find condition. For example, to search for records with values in *field* greater than 100, *criteria* has the value ">100".

Build criteria with wildcards, functions, operators, constants, and field references as you would other formulas in Approach.

Return values

Value	Description
Find	A new Find object

Usage

This method creates a Find object. When you execute the script containing the Find object, Approach creates a found set of records that match the find conditions specified in the Find object.

You can add find conditions to the Find object by calling the And and Or methods and by setting the FindSpecial property.

Approach: New method (Form class)

{button ,AL(`H_LAS_FORM_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_FORM_EXSCRIPT',1)} See example

Creates a new form.

Syntax

Set *formobject* = **New Form**(*parent,* [*maintable*])

Parameters

parent

The Document object you want to contain the form.

maintable

(Optional) A string representing the main table for the form.

Return values

Value	Description
Form	Returns a new form.

Approach: New method (LineObject class)

{button ,AL(`H_LAS_LINEOBJECT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_LINE_EXSCRIPT',1)} See example

Creates a new line.

Use LineObject object properties to set the line color, style, and width.

Syntax

Set *lineobject* = **New LineObject**(*panel*, [*page*])

Parameters

panel

The panel that contains the line. A line can be placed on the body of a form or in the footer, header, summary, or body panel of a report.

page

(Optional) For forms only. An integer (1 - 5) representing the page that contains the line.

Return values

The new object is called ObjLine. A number is added to the name if a line with that name already exists on the same panel.

For example, the name of the first line placed on the body of a form appears as follows:

ObjLine

The name of the second line appears as follows:

ObjLine1

Approach: New method (ListBox class)

{button ,AL(`H_LAS_LISTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_LISTBOX_EXSCRIPT',1)} See example

Creates a new list box.

Syntax

Set *listboxname* = **New ListBox**(*parent*, [*pagenum*])

Parameters

parent

The panel that contains the list box. Place a list box on a form or in the body panel of a report.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the list box.

Return values

The return value is the ListBox object.

Approach: New method (PicturePlus class)

{button ,AL(`H_LAS_PICTUREPLUS_CLASS;',0)} See list of classes

Creates a new PicturePlus field.

Use PicturePlus object properties to position the image within the field, or shrink, crop, or stretch the image to fit the field.

Syntax

Set *pictureplusobject* = **New PicturePlus**(*panel*, *datatable*, *datafield*, [*pagenum*])

Parameters

panel

The panel that contains the PicturePlus field. Place a PicturePlus field a form or in the body panel of a report.

datatable

A string representing the data table.

datafield

A string representing the data field.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the PicturePlus field.

Return values

The new object is called ObjPictPlus. A number is added to the name if a PicturePlus field with that name already exists on the same panel.

For example, the name of the first PicturePlus field placed on the body of a form appears as follows:

ObjPictPlus

The name of the second PicturePlus field appears as follows:

ObjPictPlus1

Approach: New method (Picture class)

{button ,AL(`H_LAS_PICTURE_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_PICTURE_EXSCRIPT',1)} See example

Creates a new picture.

Use Picture object properties to position the picture, determine which image to display, and set the shadow color.

Syntax

Set *pictureobject* = **New Picture**(*panel*, *filename*, [*pagenum*])

Parameters

panel

The panel that contains the picture. Place a picture on the body of a form, mailing label, form letter, envelope, or chart; or in the footer, header, summary, or body panel of a report.

filename

A string representing the path and filename of the image to display.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the picture.

Return values

The new object is called ObjPicture. A number is added to the name if a picture with that name already exists on the same panel.

For example, the name of the first picture placed on the body of a form appears as follows:

ObjPicture

The name of the second picture appears as follows:

ObjPicture1

Approach: New method (Query class)

{button ,AL(`H_LAS_QUERY_CLASS;',0)} See list of classes

Creates a new Query object.

Syntax

Set *queryobject* = **New Query**()

Parameters

None

Return values

Value	Description
Query object	A new instance of the Query class

Usage

Defining a query involves the following operations:

- Associating the Query object with a Connection object using the Connection property
- Specifying the find conditions for the query using the SQL property
- Specifying an entire table to retrieve using the TableName property

Approach: New method (RadioButton class)

{button ,AL(`H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_RADIOBUTTON_EXSCRIPT',1)} See example

Creates a new radio button.

Use RadioButton object properties to determine the state of a radio button, create and style labels, and set the clicked values of the radio button.

Syntax

Set *radiobuttonname* = **New RadioButton**(*panel*, [*pagenum*])

Parameters

panel

The panel that contains the radio button. Place a radio button on the body of a form or in the footer, header, summary, or body panel of a report.

Use dot notation to refer to the Panel object.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the radio button.

Return values

The new object is called ObjRadio. A number is added to the name if a radio button with that name already exists on the same panel.

For example, the name of the first radio button placed on the body of a form appears as follows:

ObjRadio

The name of the second radio button appears as follows:

ObjRadio1

Approach: New method (Rectangle class)

{button ,AL(`H_LAS_RECTANGLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_RECTANGLE_EXSCRIPT',1)} See example

Creates a new rectangle object.

Syntax

Set *rectangleobject* = **New Rectangle**(*panel*, [*pagenum*])

Parameters

panel

The panel that contains the rectangle. Place a rectangle on the body of a form, mailing label, form letter, envelope, or chart. Rectangles can also be placed in the footer, header, summary, or body panel of a report.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the rectangle.

Return values

The new object is called ObjRect. A number is added to the name if a rectangle with that name already exists on the same panel.

For example, the name of the first rectangle placed on the body of a form appears as follows:

ObjRect

The name of the second rectangle appears as follows:

ObjRect1

Approach: New method (RepeatingPanel class)

{button ,AL(`H_LAS_REPEATINGPANEL_CLASS;',0)} See list of classes

Creates a new repeating panel.

Syntax

Set *repeatingpanelobject* = **New** RepeatingPanel(*parent, table* [*pagenum*])

Parameters

parent

The Panel object that contains the repeating panel. For example, the repeating panel can be placed on the body panel of a form.

table

A string representing the table that stores the data displayed in the repeating panel on the form.

pagenum

(Optional) An integer representing the page that contains the repeating panel.

Return values

The new repeating panel is called RepeatingPanel. A number is added to the name if a repeating panel with that name already exists on the same form. For example, the name of the first repeating panel on the body of a form appears as follows:

```
RepeatingPanel
```

The name of the second repeating panel appears as follows:

```
RepeatingPanel1
```

Approach: New method (Report class)

{button ,AL(`H_LAS_REPORT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_REPORT_EXSCRIPT',1)} See example

Creates a new report.

Syntax

Set *reportobject* = **New Report**(*parent,* [*maintable*])

Parameters

parent

An object from the Document class representing where the report is placed. Use to specify which document (.APR file) is the report's parent.

maintable

(Optional) A string representing the table on which the view is based.

Return values

Value	Description
Report	Returns a new report

Approach: New method (ResultSet class)

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Creates a new ResultSet object.

Syntax

Set *resultsetobject* = **New ResultSet**()

Parameters

None

Return values

Value	Description
ResultSet	A new instance of the ResultSet class

Usage

Defining a result set involves the following operations:

- Associating the ResultSet object with a Query object using the Query property
- Creating the result set with the Execute method

Approach: New method (RoundRect class)

{button ,AL(`H_LAS_ROUNDRECT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_ROUNDRECT_EXSCRIPT',1)} See example

Creates a new rounded rectangle object.

Syntax

Set *roundrectangleobject* = **New** RoundRect(*panel*, [*pagenum*])

Parameters

panel

The panel that contains the rounded rectangle. Place a rounded rectangle on the body of a form, mailing label, form letter, envelope, or chart; or in the footer, header, summary, or body panel of a report.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the rounded rectangle.

Return values

The new object is called ObjRoundRect. A number is added to the name if a rounded rectangle with that name already exists on the same panel.

For example, the name of the first rounded rectangle placed on the body of a form appears as follows:

ObjRoundRect

The name of the second rounded rectangle appears as follows:

ObRoundRect1

Approach: New method (Sort)

{button ,AL(`H_LAS_SORT_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_NEW_SORT_EXSCRIPT',1)} See example

Creates a Sort object.

Syntax

Set *sortobject* = **New Sort** ([*field*], [*order*])

Parameters

field

(Optional) A string representing the name of the field to use in the Sort.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

order

(Optional) A long representing the order to sort the values in *field*. If you specify a value for *field*, you must specify *order*. Choose from the following values:

Value	Description
LtsSortAscending	Sorts the field values from lowest to highest and from A to Z.
LtsSortDescending	Sorts the field values from highest to lowest and from Z to A.

Return values

Value	Description
Sort	A new Sort object.

Usage

This method creates a Sort object. When you execute the script containing the Sort object, Approach sorts the found set of records or all records in the table in the order specified in the Sort object.

You can further qualify the sort condition by calling the Add method.

Approach: New method (SummaryPanel class)

{button ,AL(`H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_SUMMARYPANEL_EXSCRIPT',1)} See example

Creates a new summary panel in a report.

Syntax

Set *summarypanelobject* = **New SummaryPanel**(*parent*)

Parameters

parent

A BodyPanel object for the report view. A report view can contain a new summary panel.

Return values

The new object is called Summary. A number is added to the name of a summary if that name already exists for another summary panel on the same report.

For example, the name of the first summary panel placed on the body panel of a report appears as follows:

```
Summary
```

The name of the second summary panel appears as follows:

```
Summary1
```

Approach: New method (TextBox class)

{button ,AL(`H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_TEXTBOX_EXSCRIPT',1)} See example

Creates a new text block.

Syntax

Set *textboxobject* = **New TextBox**(*panel*, [*text*], [*pagenum*])

Parameters

panel

The panel that contains the text block. Place a text block on the body of a form, mailing label, form letter, envelope or chart; or in the footer, header, summary, or body panel of a report.

text

(Optional) A string representing text to be displayed in the text block.

pagenum

(Optional) For forms only. An integer (1 - 5) representing the page that contains the text block.

Return values

The new object is called ObjText. A number is added to the name if a text block with that name already exists on the same panel.

For example, the name of the first text block placed on the body of a form appears as follows:

ObjText

The name of the second rounded rectangle appears as follows:

ObjText1

Approach: New method (Worksheet class)

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEW_WORKSHEET_EXSCRIPT',1)} See example

Creates a new worksheet.

Syntax

Set *worksheet* **= New Worksheet**(*parent,* [*maintable*])

Parameters

parent

The document (.APR file) that contains the worksheet.

maintable

(Optional) A string representing the table for the new worksheet.

Return values

Value	Description
Worksheet	Return a new worksheet.

Approach: NextRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEXTRECORD_EXSCRIPT',1)} See example

Makes the next record in the found set the current record. Using this LotusScript command is the same as clicking the Next Record icon. If the current record is the last record, nothing happens.

Syntax

Call *docwindowobject*.**NextRecord**()

Parameters

None

Return values

None

Approach: NextRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_NEXTROW_EXSCRIPT',1)} See example

Sets the next row (record) as the current row in a result set.

Syntax

Call *resultsetobject*.**NextRow**()

or

integer = *resultsetobject*.**NextRow**()

Parameters

None

Return values

Value	Description
TRUE	The next row in the result set was set to be the current row.
FALSE	The current row did not change.

Usage

This method fails under the following conditions:

- There is no result set.
- The last row is already the active row in the result set.
- A result set is not cached.

Approach: NumColumns method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMCOLUMNS_EXSCRIPT',1)} See example

Returns the number of columns (fields) in the result set.

Syntax

integer = *resultsetobject*.**NumColumns**()

Parameters

None

Return values

Value	Description
Integer	The number of columns in the result set

Approach: NumParameters method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Returns the number of parameters in an SQL statement.

Syntax

integer = *resultsetobject*.**NumParameters**()

Parameters

None

Return values

Value	Description
Integer	The number of parameters in an SQL statement

Usage

In SQL, a parameter is a variable that is used in an SQL statement. The parameter name appears in the SQL statement surrounded by question marks (?). Use this method to return the number of parameters found in the SQL statement used to create the result set.

Approach: NumRows method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMROWS_EXSCRIPT',1)} See example

Returns the number of rows (records) in the result set.

Syntax

long = *resultsetobject*.**NumRows**()

Parameters

None

Return values

Value	Description
Long	The number of rows in the result set

Approach: OpenDocument method

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Opens the specified document (.APR file) or table (database file).

Syntax

document = *applicationobject*.**OpenDocument**(*filename, filepath*, [*doctype*], [*password*], [*openro*], [*makevisible*])

Parameters

filename

A string representing the file name of a document or table.

filepath

A variant representing the path of the file specified in *filename*.

doctype

(Optional) A string representing the file type of the document. Choose from the following values:

Value	Description
Lotus Approach (*.APR, *.VEW, *.APT)	Approach file for storing views, or consolidated Approach data and views
SmartMaster (*.MPR)	Approach SmartMaster Application
dBASE IV (*.DBF)	Borland dBASE IV
dBASE IIII+ (*.DBF)	Borland dBASE III+
Lotus Notes Workspace, Server, Local (*)	Lotus Notes
FoxPro (*.DBF)	FoxPro
Paradox (*.DB)	Borland Paradox
Query (*.QRY)	Approach query file, for SQL tables
Text(*.TXT)	Delimited or fixed-length text
Excel (*.XLS)	Microsoft Excel
Lotus 1-2-3 (*.123, *.WK*)	1-2-3 workbook

password

(Optional) A string representing the password required to open the document.

openro

(Optional) An integer representing whether to open the document or table as read-only. Choose from the following values:

Value	Description
TRUE	Approach opens the document as read-only.
FALSE	Approach opens the document as read-write.

makevisible

(Optional) An integer representing whether to make the document current after it is open. Choose from the following values:

Value	Description
TRUE	Approach opens the document and immediately changes the focus to the document.

FALSE	Approach opens the document but keeps the focus on another document.

Return values

Value	Description
Document	The specified Document object.

Usage

This method opens an existing .APR file or database file.

If you specify a database as the file to open, Approach creates a new file (.APR) with a default form and worksheet.

Caution If you select a file with a .DBF or .DB file extension, the program creates a new Approach file for it, even if an Approach file associated with the .DBF or .DB file already exists. If you save this new Approach file, you will write over the existing one.

Approach: Options method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Selects the record-locking behavior of the ResultSet object.

Syntax

integer = *resultsetobject*.**Options**(*option*)

Parameters

option

An integer indicating the record-locking behavior of the ResultSet object. Choose from the following options:

Value	Description
DBOpt_OPTIMISTIC	Allows all users to save their changes to a record (optimistic record locking).
DBOpt_OVERRIDE	Updates records regardless of locks. This option requires support from the data source.
DBOpt_PESSIMISTIC	Allows only the first user to save changes to a record (full record locking).

Return values

Value	Description
TRUE	The record-locking behavior was successfully set to the new mode.
FALSE	The record-locking behavior was not changed.

Approach: Or method

{button ,AL(`H_LAS_FIND_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_OR_EXSCRIPT',1)} See example

Adds a find condition to an existing Find object.

This new find condition is in an OR relationship with the previously added find condition.

Syntax

Call *findobject*.**Or**(*field*,*criteria*)

Parameters

field

A string representing the name of the field to use in this find condition.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

criteria

A string representing the find condition.

Build criteria with wildcards, functions, operators, constants, and field references as you would other formulas in Approach.

Return values

None

Usage

This method adds a find condition to an existing Find object. The find conditions are in an OR relationship, which means that only one of the conditions must be met by a record for that record to be included in the found set.

Approach: Paste method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Paste the contents of the Clipboard to the position you most recently selected.

Syntax

Call *docwindowobject*.**Paste()**

Parameters

None

Return values

None

Usage

Use the Paste method after using the Cut or Copy methods.

Approach: PrevRecord method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_PREVRECORD_EXSCRIPT',1)} See example

Makes the previous record in the found set the current record.

Using this LotusScript method is the same as clicking the Previous Record icon. If the current record is the first record, nothing happens.

Syntax

Call *docwindowobject*.**PrevRecord**()

Parameters

None

Return values

None

Approach: PrevRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_PREVROW_EXSCRIPT',1)} See example

Sets the previous row (record) in the result set as the current row.

Syntax

integer = *resultsetobject*.**PreviousRow**

Parameters

None

Return values

Value	Description
TRUE	The previous row in the result set was set to be the current row.
FALSE	The current row did not change.

Usage

This method fails if the previous row is not accessible. This may happen under the following conditions:

- You are at the first row.
- There is no valid result set to scroll through.
- A result set is not cached.
- The cache is limited and the ODBC driver does not support cursor or result navigation.

Approach: PrintPreview method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Switches Approach into Print Preview .

Syntax

integer = *docwindowobject*.**PrintPreview()**

Parameters

None

Return values

Value	Description
TRUE	Approach successfully switched to Print Preview.
FALSE	Approach failed to switch to Print Preview.

Usage

Use this method to toggle the Approach environment between Print Preview and Browse.

Approach: Print method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Sends the specified pages of the current view to the printer you specify.

Syntax

Call *docwindowobject*.**Print**([*from*], [*to*], [*numcopies*])

Parameters

from

(Optional) Long representing the number of the first page to print. The default value is page 1.

to

(Optional) Long representing the number of the last page to print. The default value is the last page of the view.

numcopies

(Optional) Integer representing the number of copies to print. The default value is 1 copy of each page.

Return values

None

Usage

Switch to the view you want to print using the ActiveView property; then call the Print method.

Approach: Quit method

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

{button ,AL(`H_las_QUIT_EXSCRIPT',1)} See example

Closes the current Approach executable (.EXE file).

Syntax

Call *applicationobject*.**Quit**([*savechanges*])

Parameters

savechanges

(Optional) A variant representing whether to save changes made to the document (.APR file).

Return values

Value	Description
TRUE	All documents (.APR files) are saved.
FALSE	The documents are not saved.

Approach: Refresh method

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_REFRESH_EXSCRIPT',1)} See example

For DocWindow objects:

- Updates the data onscreen to match the source table (database)
- Enters changes to the database
- Displays new records in the found set or sort order, as appropriate

For display element objects, Refresh redraws the display element.

Syntax

Call *docwindowobject*.**Refresh**()

or

Call *displayelementobject*.**Refresh**()

Parameters

None

Return values

Value	Description
TRUE	The display element or document window is updated.
FALSE	The display element or document window is not updated.

Usage

If several users are making changes to a table at the same time, the changes do not always appear instantly on each user's screen. Use Refresh from a DocWindow object to update a local copy of the data in the source table (database).

Approach: RemoveColumn method

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Removes the specified column from the worksheet. If no column is specified, the current column is removed.

Syntax

integer = *worksheetobject*.**RemoveColumn**(*column*)

Parameters

column

A string representing the header of the column. The header, or label, may differ from the column's field name.

Return values

Value	Description
TRUE	The specified column was removed from the worksheet.
FALSE	The specified column was not removed from the worksheet.

Approach: RemoveListItem method

{button ,AL(`H_LAS_LISTBOX_CLASS;',0)} See list of classes

Deletes the specified item from the list for a list box.

Syntax

integer = *listboxobject*.**RemoveListItem**(*item*)

Parameters

item

An integer representing an item in the list box.

Return values

Value	Description
TRUE	Approach deleted the item successfully.
FALSE	Approach failed to remove the item, possibly because the item specified was not in the list.

Usage

Use this method to control the values allowed for data entry in a field. For example, you can delete items from a list box in response to other data entered by the user.

Approach: Remove method

{button ,AL(`H_LAS_COLLECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_REMOVE_EXSCRIPT',1)} See example

Removes an object from the current collection at the specified position.

Syntax

object = *collectionobject*.**Remove**(*position*)

Parameters

position

An integer representing a specified position in the collection.

Return values

Value	Description
Object	Returns the object that was removed from the collection.

Approach: Repaint method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_REPAINT_EXSCRIPT',1)} See example

Repaints the document window.

Syntax

Call *docwindowobject*.**Repaint**()

Parameters

None

Return values

None

Approach: ReplaceWithResultSet method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_REPLACEWITHRESULTSET_EXSCRIPT',1)} See example

Replaces a table associated with a document (.APR file) with a result set table.

Syntax

integer = *tableobject*.**ReplaceWithResultSet**(*resultsetobject,* [*fieldpairs*])

Parameters

resultsetobject

A ResultSet object that contains the data that replaces the original table associated with the .APR file.

fieldpairs

(Optional) An array describing the mapping of fields between the original table and the result set. If you do not specify *fieldpairs* and the fields in the result set do not exactly match the fields in the original table, Approach opens the Mapping dialog box to allow the user to match the fields between tables.

Construct the array in two dimensions with the first dimension as large as the number of fields in the original table. List each field name from the original table in the first element of each pair in the array; list the field name from the result set in the second element. For example, in the field pair array MyMap, define the mapping for the field names as follows:

```
Dim MyMap(1 to 5, 1 to 2) As String
MyMap(1,1) = "First Name"      ' Field from original table
MyMap(1,2) = "FIRST"           ' Field from result set
MyMap(2,1) = "Last Name"       ' Field from original table
MyMap(2,2) = "LAST"            ' Field from result set
. . .
MyMap(5,1) = "Postal Code"     ' Field from original table
MyMap(5,2) = "PCODE"           ' Field from result set
```

Return values

Value	Description
TRUE	Approach successfully replaced the specified Table object with the ResultSet object.
FALSE	Approach failed to replace the specified Table object with the ResultSet object.

Usage

This method lets you replace a table associated with the document (.APR file). After calling ReplaceWithResultSet, the document retrieves data as defined by the new table (the result set). The original table is not deleted or altered, but is no longer associated with the document. The new table does not reflect subsequent changes to the Query object associated with the result set.

Create a ResultSet object from a table not associated with the document using the Connection, Query, and ResultSet classes. Create a ResultSet object from a table associated with the document using the CreateResultSet method.

Specify the Table object by using the Tables property of the Document object.

Approach: Replicate method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Updates copies of the same Notes table (database file) located on two different servers, or on a server and a local machine.

Syntax

Call *docwindowobject*.**Replicate**(*filename*, *server*, [*exchangereadmarks*], [*receivedocuments*], [*senddocuments*], [*replicatetemplates*])

Parameters

filename

A string representing the full path name of the Notes table on the server.

server

A string representing the server name.

exchangereadmarks

(Optional) An integer indicating whether the status of each record (Notes document) in the database should be updated during replication. Read marks indicate whether a record has been opened. Choose from the following values:

Value	Description
TRUE	Approach updates the read marks during replication.
FALSE	(Default) Approach does not update the read marks.

receivedocuments

(Optional) An integer indicating whether data is copied from the server database to the local database. Choose from the following values:

Value	Description
TRUE	Approach updates the local database with data from the server copy.
FALSE	Approach does not update the local database with data from the server copy.

senddocuments

(Optional) An integer indicating whether data is copied from the local database to the server database. Choose from the following values:

Value	Description
TRUE	Approach updates the server database with data from the local copy.
FALSE	Approach does not update the server database with data from the local copy.

replicatetemplates

(Optional) An integer indicating whether the Notes template information is updated during the replication. Choose from the following values:

Value	Description
TRUE	Approach updates the template during replication.
FALSE	Approach does not update the template during replication.

Return values
None

Usage
This method automates Notes database replication.

Approach: Restore method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;H_LAS_DOCWINDOW_CLASS;H_LAS_WINDOW_CLASS;', 0)} See list of classes

{button ,AL(`H_las_RESTORE_EXSCRIPT',1)} See example

Restores a minimized or maximized application or document window, and makes it the active window.

Using this LotusScript method is the same as clicking the Restore icon.

Syntax

Call *applicationwindowobject*.**Restore**()

or

Call *docwindowobject*.**Restore**()

Parameters

None

Return values

None

Approach: RunProcedure method

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_RUNPROCEDURE_EXSCRIPT',1)} See example

Executes an existing sub.

Syntax

applicationobject.**RunProcedure**(*procedurename, procedurearg*)

Parameters

procedurename

A string representing the name of the global sub to be executed.

procedurearg

A string representing the argument of the sub.

Return values

Value	Description
TRUE	The sub was executed.
FALSE	The sub was not executed.

Usage

This method allows you to run a global sub from another product through OLE automation. For example, you can run an Approach script by creating an Approach application inside a 1-2-3 script, opening an existing Approach .APR file (document), and then calling RunProcedure.

RunProcedure cannot execute a function or a sub with more than one argument.

Approach: SameColor method

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

Compares RGB values to see if they are the same. You can also compare colors from different products.

Syntax

value = *colorobject*.**SameColor**(*othercolor*)

Parameters

othercolor

The color object you want to compare to the current color object.

Return values

Value	Description
TRUE	The stored RGB values are the same.
FALSE	The stored RGB values are not the same.

Approach: SaveChanges method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Saves all unsaved design modifications to the specified document (.APR file).

Syntax

Call *docwindowobject*.**SaveChanges()**

Parameters

None

Return values

None

Usage

This method saves changes to a document that has already been saved once, so it has a file name.

Approach: SaveViewAsHTML method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Creates an HTML file representing the current view.

Data type

String

Syntax

string = *DocWindowobject*.**SaveViewAsHTML**([*outputfile*])

Parameters

outputfile

(Optional) A string representing the full path name of the output HTML file.

Return values

Value	Description
string	Full path name of the resulting HTML file.

Usage

Use this view to convert a view to HTML format. If the current view is a form, the resulting HTML form appears without data, allowing viewers to enter data in the fields on the form. If the current view is another type of view, the data are represented in the resulting HTML document.

Approach: SelectAll method

{button ,AL(`;H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Select all text in a field box or all cells in a worksheet.

Syntax

Call *docwindowobject*.**SelectAll()**

Parameters

None

Return values

None

Usage

This method automates the Edit - Select All menu command.

The Select All command has different functions in different contexts. For example, when the cursor is in a field box on a form, Select All selects the contents of the field box.

In a worksheet, Select All selects all records in the found set.

Approach: SendToBack method

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SENDTOBACK_EXSCRIPT',1)} See example

Places a display element behind all overlapping display elements.

Syntax

displayelement object.**SendToBack**

Parameters

None

Return values

None

Usage

Arrange display elements so they do not hide other display elements.

For example, you can have a circle around a group of radio buttons. For users to be able to click the buttons, however, the circle must be behind the radio buttons.

Approach: SetAt method

{button ,AL(`H_LAS_COLLECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETAT_EXSCRIPT',1)} See example

Replaces the object at the specified position with a specified object.

Syntax

variant = *collectionobject*.**SetAt**(*position, object*)

Parameters

position

An integer representing an object in the collection.

object

A variant representing the object with which you want to replace the object in *position*.

Return values

Value	Description
Variant	Returns the object previously at *position*.

Usage

This method signals a bounds error if *position* is less than one, or greater than Count (which indicates that the size of the collection has not changed). SetAt is normally accessed via indexing.

Approach: SetCellFocus method

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETCELLFOCUS_EXSCRIPT',1)} See example

Selects the cell in the current row and specified column.

Syntax

integer = *worksheetobject*.**SetCellFocus**(*columnname*)

Parameters

columnname

A string representing the header of the column. The header, or label, may differ from the column's field name.

Return values

Value	Description
TRUE	The specified cell was selected.
FALSE	The specified cell was not selected.

Approach: SetFieldList method

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;H_LAS_LISTBOX_CLASS;',0)} See list of classes

Determines the items displayed in a drop-down box, a field box and list, or a list box.

Syntax

integer = *field*.**SetFieldList**(*fielddata*, [*fielddesc*], [*fieldfilterfrom*],[*fieldfilterto*])

Parameters

fielddata

The name of the field that stores the items for the list.

If there is more than one table joined in the .APR file, precede the field name with the table name. For example, the field Cost is in the table Products in an .APR file that also uses the table Orders. You must specify the field name as follows:

```
"Products.Cost"
```

The field and table names are not case sensitive.

fielddesc

(Optional)The name of the field that stores the descriptions of the items in the list. If specified, the contents of this field are displayed to users instead of the data in *fielddata*.

If there is more than one table joined in the .APR file, specify the table name as described above.

fieldfilterfrom

(Optional) The name of the field the items are filtered from.

If there is more than one table joined in the .APR file, specify the table name as described above.

fieldfilterto

(Optional) The name of the field the items are filtered to.

If there is more than one table joined in the .APR file, specify the table name as described above.

Return values

Value	Description
TRUE	The field list is set for the field.
FALSE	The field list is not set for the field.

Approach: SetFocus method

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_DISPLAY_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETFOCUS_EXSCRIPT',1)} See example

Selects the specified display element, as if a user had tabbed into or clicked the display element.

Syntax

displayelementobject.**SetFocus**

Parameters

None

Return values

None

Approach: SetFurigana method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

This method is not used in this version of Approach.

Approach: SetIME method

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

This method is not used in this version of Approach.

Approach: SetList method

{button ,AL(`H_LAS_DROPDOWNBOX_CLASS;H_LAS_LISTBOX_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETLIST_EXSCRIPT',1)} See example

Displays the items in a drop-down box, field box and list, or list box from the data in a specified array.

Syntax

integer = *field*.**SetList**(*stringarray)*

Parameters

stringarray

An array of strings that display as items in a drop-down box, field box and list, or list box.

Return values

Value	Description
TRUE	The items are set for the field.
FALSE	The items are not set for the field.

Approach: SetParameter method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

Modifies a parameter value.

Syntax

integer = *resultsetobject*.**SetParameter**(*parameter*, *value*)

Parameters

parameter

A string representing the name of the parameter.

Alternatively, you can use an integer representing the ordinal position of the parameter, if the name is unknown. The first parameter in the SQL statement has an index of 1, the second has an index of 2, and so on.

value

A string representing the new parameter value. Use single quotation marks in *value* when setting the value of string parameters.

Return values

Value	Description
TRUE	The replacement value for the parameter was set.
FALSE	The replacement value for the parameter was not set.

Usage

This method fails when the parameter name or index is not found.

A parameter is a variable used in an SQL statement. The parameter name appears in the SQL statement surrounded by question marks (?). Use this method to change the value used in the SQL statement.

Approach: SetPicture method

{button ,AL(`H_LAS_PICTURE_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETPICTURE_EXSCRIPT',1)} See example

Determines the image to be displayed.

Syntax

integer = *pictureobject*.**SetPicture**(*filename)*

Parameters

filename

A string representing the location and filename for the image.

Return values

Value	Description
TRUE	The specified picture was placed in the picture field.
FALSE	The specified picture was not placed in the picture field.

Approach: SetRGB method

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETRGB_EXSCRIPT',1)} See example

Sets one of the predefined color constants as the color to use for the Color object.

Syntax

integer = *colorobject*.**SetRGB**(*rgbvalue*)

Parameters

rgbvalue

A long representing an RGB value for a Color object. Choose from predefined color constants.

Return values

Value	Description
TRUE	The RGB components are set.
FALSE	The RGB components are not set.

Approach: SetState method

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETSTATE_EXSCRIPT',1)} See example

Sets the default state of a check box or radio button. A check box is either checked or unchecked. A radio button is either on or off.

Syntax

integer = *displayelementobject*.**SetState**(*value*)

Parameters

value

An integer representing the state of the check box or radio button.

Return values

Value	Description
TRUE	Set the default value of the check box or radio button to on (checked).
FALSE	Set the default value of the check box or radio button to off (unchecked).

Approach: SetText method

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Enters text in the cell of the current row of the specified column. If no column is specified, the current column is used.

Syntax

integer = *worksheetobject*.**SetText**(*text,* [*columnname*])

Parameters

text

A string representing the text to be entered in the current row of the specified column.

columnname

(Optional) A string representing the header of the column. The header, or label, may differ from the column's field name.

Return values

Value	Description
TRUE	The text for the current row of the specified column was set.
FALSE	The text for the current row of the specified column was not set.

Approach: SetValue method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_SETVALUE_EXSCRIPT',1)} See example

Modifies the data in the current row (record) and specified column (field) of a result set.

Syntax

integer = *resultsetobject*.**SetValue**(*column*, *value*)

Parameters

column

A string representing the name of a column (field) in a result set.

Alternatively, you can use an integer representing the ID value of the column, if the name is unknown. The first parameter in the SQL statement has an index of 1, the second has an index of 2, and so on.

value

A variant representing the value added to the current row and specified column of the result set.

Return values

Value	Description
TRUE	The new value was successfully placed in the result set.
FALSE	The new value failed to be placed in the result set, probably because the column or result set specified does not exist.

Usage

The current row and specified column of the result set identify a cell in the source table.

You must call the UpdateRow method to save the change in the result set.

The new value is automatically converted to the native data type specified for the column.

Approach: TabNext method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABNEXT_EXSCRIPT',1)} See example

Tabs to the next display element in the tab order .

Using this LotusScript method is the same as pressing TAB.

Syntax

Call *docwindowobject*.**TabNext**()

Parameters

None

Return values

None

Usage

Use this method to move the cursor to the next display element.

To move to:

- the previous display element, use the TabPrev method
- a specific display element, use the TabTo method

Approach: TabPrev method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABPREV_EXSCRIPT',1)} See example

Tabs to the previous object in the tab order .

Using this LotusScript command is the same as pressing SHIFT+TAB.

Syntax

Call *docwindowobject*.**TabPrev**()

Parameters

None

Return values

None

Usage

Use this method to move the cursor to the previous display element.

To move to:

- the next display element, use the TabNext method
- a specific display element, use the TabTo method

Approach: TabTo method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Moves the focus to the specified display element in the current view.

Syntax

Call *docwindowobject*.**TabTo** (*tabposition*)

Parameters

tabposition

An integer representing the position in the tab order of the object you want to move to. The *tabposition* corresponds to the value of the TabOrder property of the display element in the view.

Return values

None

Usage

This method allows you to control the order that users enter data.

For example, you can use values entered by the user to control what fields are filled in and in what order.

To move to:

- the previous display element, use the TabPrev method
- the next display element, use the TabNext method

Approach: Tile method

{button ,AL(`H_LAS_APPLICATIONWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_TILE_EXSCRIPT',1)} See example

Tiles all open windows. Using this LotusScript command is the same as if you chose Window - Tile.

Syntax

Call *applicationwindowobject*.**Tile**()

Parameters

None

Return values

None

Approach: Transactions method

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

Sets the data transaction mode for a connection when the AutoCommit property is FALSE.

Syntax

integer = *connectionobject*.**Transactions**(*mode*)

Parameters

mode

An integer representing the way Approach writes data to a table. Choose from the following values:

Value	Description
AprTransactionCommit	Any changes made to data since the previous transaction are committed to the table when you call UpdateRow.
AprTransactionRollback	Any changes made to data since the previous transaction are discarded.

Return values

Value	Description
TRUE	Approach successfully changed the transaction mode.
FALSE	The table does not support the specified transaction mode.

Usage

Use the Transactions method to access SQL transaction controls on a server database.

When the AutoCommit property of the Connection object is enabled (set to TRUE), the Transactions method is disabled.

After you rollback changes, call the Refresh method to update the data on the screen.

Approach: Unsort method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Returns the current found set to the default sort order specified in File - User Setup - Approach Preferences.

Syntax

Call *docwindowobject*.**Unsort()**

Parameters

None

Return values

None

Usage

To set the sort order for the records in which they appear by default in Browse , choose File - User Setup - Approach Preferences.

Approach: UpdateRow method

{button ,AL(`H_LAS_RESULTSET_CLASS;',0)} See list of classes

{button ,AL(`H_las_UPDATEROW_EXSCRIPT',1)} See example

Updates the current row (record) in the result set with any changes.

Syntax

Call *resultsetobject*.**UpdateRow**()

or

integer = *resultsetobject*.**UpdateRow**()

Parameters

None

Return values

Value	Description
TRUE	The current row in the result set was updated.
FALSE	The current row in the result set was not updated.

Usage

You must call this method to save changes you make to values in a result set.

Updating a table may also depend on the Connection object's settings for commiting records. See the Connection class AutoCommit property.

Approach: Zoom method

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Enlarges or reduces the display of the current view in Print Preview.

Syntax

integer = *docwindowobject*.**Zoom**(*percentage*)

Parameters

percentage

An integer representing the new display as a percentage. Choose from 25, 50, 75, 85, 100 and 200.

Return values

Value	Description
TRUE	The view successfully zoomed to the specified scale.
FALSE	The view failed to zoom successfully.

Usage

Use the Zoom method to change the display scale when the view is in PrintPreview.

Approach LotusScript A-Z

A

B

C

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

X

(None)

Y

Z

Approach LotusScript Classes A-Z

A

- Application class
- ApplicationWindow class

B

- Background class
- BaseCollection class
- BodyPanel class
- Border class
- Button class

C

- ChartView class
- CheckBox class
- Collection class
- Color class
- Connection class
- Crosstab class

D

Display class
Document class
DocWindow class
DropdownBox class

E

Ellipse class
Envelope class

F

FieldBox class
Find class
FindDistinct class
FindDuplicate class
FindTopLowest class
Font class
Form class
FormLetter class

G

(None)

H

HeaderFooterPanel class

I

(None)

J

(None)

K

(None)

L

LineObject class
LineStyle class
ListBox class

M

MailingLabels class

N

(None)

O

OLEObject class

P

Panel class
Picture class
PicturePlus class

Q

Query class

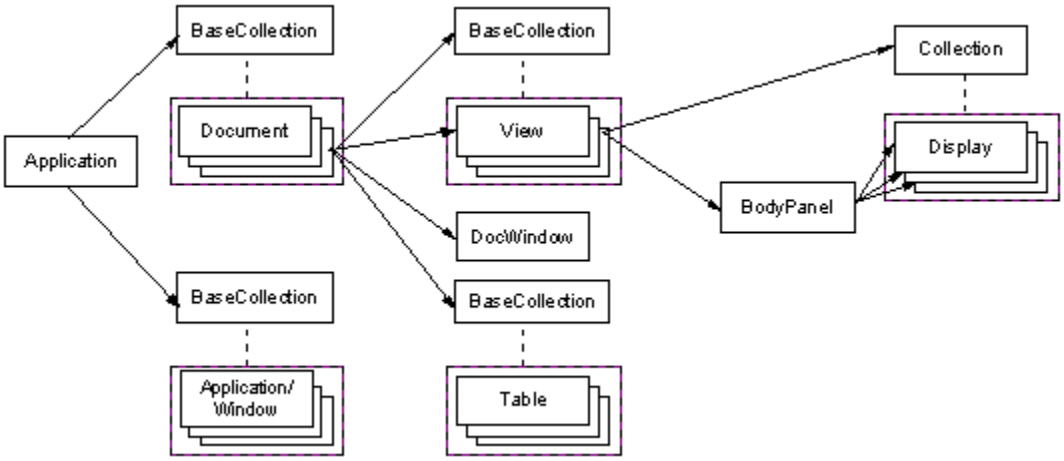
R

RadioButton class

Approach LotusScript Class Hierarchy

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.

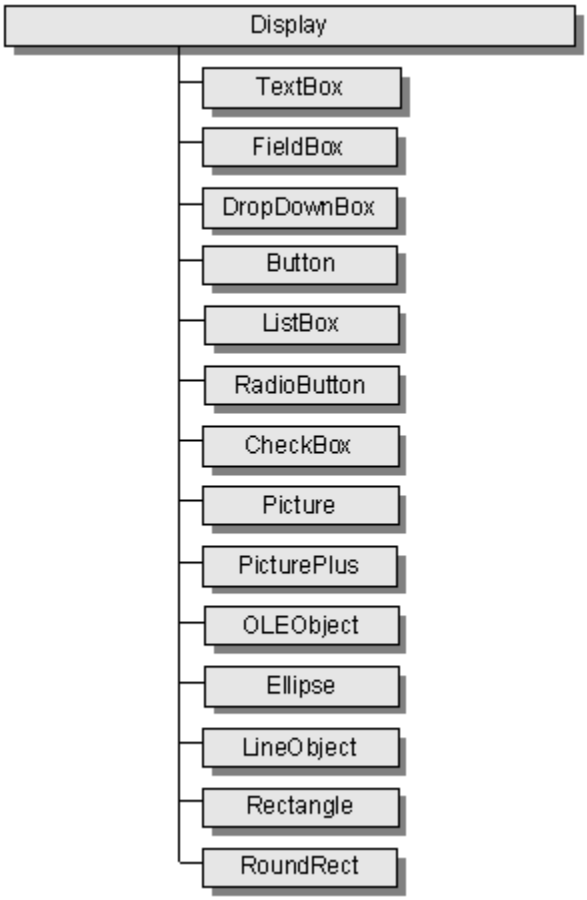
The following diagram illustrates the containment relationships of classes and objects.

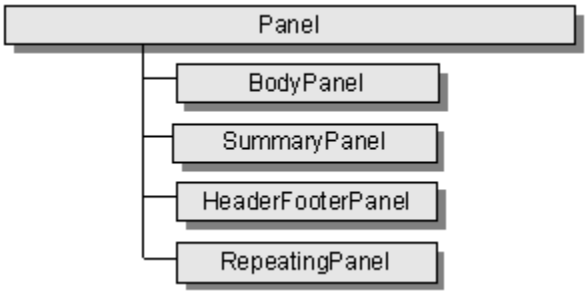


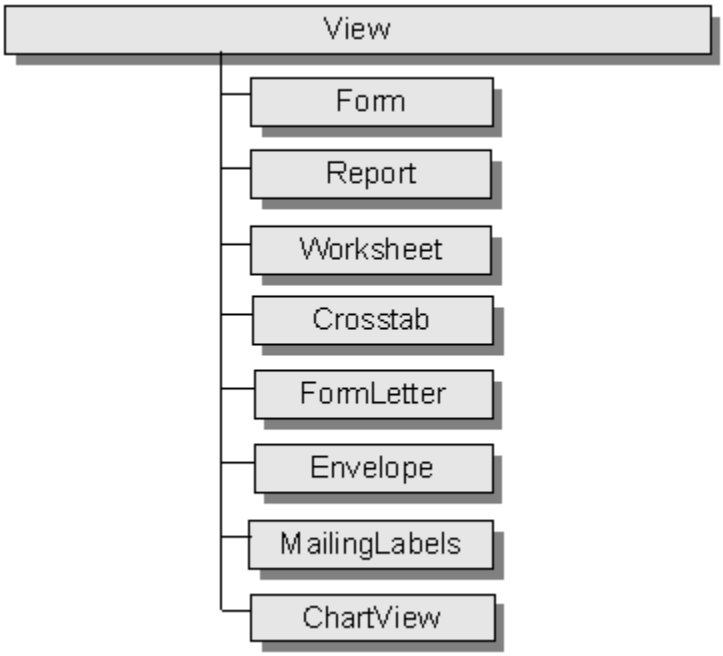
Approach provides some classes, called abstract classes, that exist only to create other classes, called derived classes. You cannot create an instance (object) of an abstract class.

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 diagrams show the Approach class hierarchy, including the abstract classes (in italics) and the derived classes that inherit from them.


Display
TextBox
FieldBox
DropDownBox
Button
ListBox
RadioButton
CheckBox
Picture
PicturePlus
OLEObject
Ellipse
LineObject
Rectangle
RoundRect


Panel
BodyPanel
SummaryPanel
HeaderFooterPanel
RepeatingPanel


View
Form
Report
Worksheet
Crosstab
FormLetter
Envelope
MailingLabels
ChartView

Approach LotusScript Constants

Approach scripts use the following constants to specify menu commands, colors, and database connection errors available in Approach.

Menu command	Constant
About Approach...	IDM_ABOUT
Actual Size	IDM_ACTUAL
Add Column	IDM_ADDCOLUMN
Approach File Info...	IDM_APRINFO
Ascending Sort	IDM_SORTUP
Browse	IDM_BROWSE
Chart Crosstab	IDM_CHART
Clear	IDM_CLEAR
Close	IDM_CLOSE
Close and Disconnect from Server	IDM_CLOSEANDDISC
Copy	IDM_COPY
Copy to File...	IDM_COPYTOFILE
Create Chart...	IDM_NEWCHART
Create CrossTab...	IDM_NEWXTAB
Create Form...	IDM_NEWFORM
Create Form Letter...	IDM_NEWLETTER
Create Mailing Labels...	IDM_NEWLABELS
Create Object... **Note** Create Object is not supported under OS/2.	IDM_INSOBJECT
Create Control (OLE)... **Note** Create Control is not supported under OS/2.	IDM_INSCONTROL
Create Report...	IDM_NEWREPORT
Create Worksheet...	IDM_NEWWORKSHEET
Customize Menus...	IDM_MENUS
Cut	IDM_CUT
Define Sort...	IDM_SORT
Delete File...	IDM_DELETEFILE
Delete Found Set	IDM_DELETEFOUND
Delete Record	IDM_DELETERECORD
Descending Sort	IDM_SORTDOWN
Design	IDM_DESIGN
Duplicate Record	IDM_DUPLICATE
Edit Column Label	IDM_EDITLABEL
Edit OLE Object... **Note** Edit OLE Object is not supported under OS/2.	IDM_EDITOLE
Export Data...	IDM_EXPORT
Exit	IDM_EXIT
Fast Format	IDM_FASTFORMAT
Field Definition...	IDM_DEFINE

Fill Field...	IDM_FILL
Find	IDM_FIND
Find Assistant...	IDM_FINDASSISTANT
Find Again	IDM_FINDAGAIN
Find Assistant... (for finding duplicate or distinct records)	IDM_FINDSPECIAL
First Record	IDM_FIRST
Help Contents	IDM_HELPINDEX
Help Customer Support	IDM_HELPSUPPORT
Help Search...	IDM_HELPSEARCH
Help Using Help	IDM_HELPHELP
Hide Record	IDM_HIDE
Import Data...	IDM_IMPORT
Insert Today's Date	IDM_DATE
Insert Current Time	IDM_TIME
Insert Previous Value	IDM_PREVDATA
Join...	IDM_JOIN
Last Record	IDM_LAST
Links... **Note** Links is not supported under OS/2.	IDM_LINKS
Macros...	IDM_MACROS
Named Styles...	IDM_STYLES
New...	IDM_NEW
New Record	IDM_NEWREC
Next Record	IDM_NEXT
Notes New Notes Replica...	IDM_NEWREPLICA
Notes Replicate with Notes Server...	IDM_REPLICATE
Object Properties	IDM_INFOBOX
Open...	IDM_OPEN
Preferences...	IDM_OPTIONS
Paste	IDM_PASTE
Paste from File...	IDM_PASTEFILE
Paste Special...	IDM_PASTELINK
Preview	IDM_PREVIEW
Previous Record	IDM_PREV
Print...	IDM_PRINT
Print Setup...	IDM_PRINTSETUP
Refresh	IDM_REFRESH
Replace	IDM_REPLACE
Save As...	IDM_SAVEAS
Save Approach File	IDM_SAVE
Select All	IDM_SELECTALL
Select Cells Only	IDM_SELECTCELLS
Select Header Only	IDM_SELECTLABEL
Send Mail...	IDM_SENDMAIL

Show Action Bar	IDM_SHOW_ACTION_BAR
Show All	IDM_SHOWALL
Show Data	IDM_SHOWDATA
Show Internet Tools	IDM_SHOW_INTERNET_TOOLS
Show SmartIcons	IDM_SHOWICONS
Show Status Bar	IDM_SHOWSTATUS
Show View Tabs	IDM_SHOWTABS
SmartIcons...	IDM_SMARTICONS
Spell Check...	IDM_SPELLCHECK
Summarize Columns	IDM_ADDROW
Summarize Rows	IDM_ADDCOLUMN
Tile Left-Right	IDM_TILE_LEFT_RIGHT
Tile Top-Bottom	IDM_TILE_TOP_BOTTOM
Undo	IDM_UNDO
View Properties	IDM_INFOBOX_VIEW
Zoom In	IDM_ZOOMIN
Zoom Out	IDM_ZOOMOUT

Color constants	**RGB contributions**
COLOR_WHITE	0x02FFFFFFL
COLOR_VANILLA	0x02FFEFCEL
COLOR_PARCHMENT	0x02FFFFC2L
COLOR_IVORY	0x02FFFFD0L
COLOR_PALE_GREEN	0x02E0FFBFL
COLOR_SEA_MIST	0x02E0FFDFL
COLOR_ICE_BLUE	0x02E0FFFFL
COLOR_POWDER_BLUE	0x02C2EFFFL
COLOR_ARCTIC_BLUE	0x02E0F1FFL
COLOR_LILAC_MIST	0x02E0E0FFL
COLOR_PURPLE_WASH	0x02E8E0FFL
COLOR_VIOLET_FROST	0x02F1E0FFL
COLOR_SEASHELL	0x02FFE0FFL
COLOR_ROSE_PEARL	0x02FFE0F5L
COLOR_PALE_CHERRY	0x02FFE0E6L
COLOR_WHITE	0x02FFFFFFL
COLOR_BLUSH	0x02FFE1DCL
COLOR_SAND	0x02FFE1B0L
COLOR_LIGHT_YELLOW	0x02FFFF80L
COLOR_HONEYDEW	0x02F1F1B4L
COLOR_CELERY	0x02C2FF91L
COLOR_PALE_AQUA	0x02C1FFD5L
COLOR_PALE_BLUE	0x02C1FFFFL
COLOR_CRYSTAL_BLUE	0x02A1E2FFL
COLOR_LT_CORNFLOWER	0x02C0E1FFL
COLOR_PALE_LAVENDER	0x02BFBFFFL
COLOR_GRAPE_FIZZ	0x02D2BFFFL

COLOR_PALE_PLUM	0x02E1BFFFL
COLOR_PALE_PINK	0x02FFC1FDL
COLOR_PALE_ROSE	0x02FFC0E4L
COLOR_ROSE_QUARTZ	0x02FFC0CEL
COLOR_3_GRAY	0x02F7F7F7L
COLOR_RED_SAND	0x02FFC0B6L
COLOR_BUFF	0x02FFC281L
COLOR_LEMON	0x02FFFF35L
COLOR_PALE_LEMON_LIME	0x02F1F180L
COLOR_MINT_GREEN	0x0280FF80L
COLOR_PASTEL_GREEN	0x0282FFCAL
COLOR_PASTEL_BLUE	0x0280FFFFL
COLOR_SAPPHIRE	0x0282E0FFL
COLOR_CORNFLOWER	0x0282C0FFL
COLOR_LIGHT_LAVENDER	0x029F9FFFL
COLOR_PALE_PURPLE	0x02C29FFFL
COLOR_LIGHT_ORCHID	0x02E29FFFL
COLOR_PINK_ORCHID	0x02FF9FFFL
COLOR_APPLE_BLOSSOM	0x02FF9FCFL
COLOR_PINK_CORAL	0x02FF9FA9L
COLOR_6_GRAY	0x02EFEFEFL
COLOR_LIGHT_SALMON	0x02FF9F9FL
COLOR_LIGHT_PEACH	0x02FF9F71L
COLOR_YELLOW	0x02FFFF00L
COLOR_AVOCADO	0x02E0E074L
COLOR_LEAF_GREEN	0x0241FF32L
COLOR_LIGHT_AQUA	0x0242FFC7L
COLOR_LT_TURQUOISE	0x0242FFFFL
COLOR_LIGHT_CERULEAN	0x0200BFFFL
COLOR_AZURE	0x025291EFL
COLOR_LAVENDER	0x028080FFL
COLOR_LIGHT_PURPLE	0x02C082FFL
COLOR_DUSTY_VIOLET	0x02E081FFL
COLOR_PINK	0x02FF7FFFL
COLOR_PASTEL_PINK	0x02FF82C2L
COLOR_PASTEL_RED	0x02FF82A0L
COLOR_12_GRAY	0x02E1E1E1L
COLOR_SALMON	0x02FF8080L
COLOR_PEACH	0x02FF8141L
COLOR_MUSTARD	0x02FFE118L
COLOR_LEMON_LIME	0x02E1E140L
COLOR_NEON_GREEN	0x0200FF00L
COLOR_AQUA	0x0200FFB2L
COLOR_TURQUOISE	0x0200FFFFL
COLOR_CERULEAN	0x0200A1E0L

COLOR_WEDGEWOOD	0x022181FFL
COLOR_HEATHER	0x026181FFL
COLOR_PURPLE_HAZE	0x02A160FFL
COLOR_ORCHID	0x02C062FFL
COLOR_FLAMINGO	0x02FF5FFFL
COLOR_CHERRY_PINK	0x02FF60AFL
COLOR_RED_CORAL	0x02FF6088L
COLOR_18_GRAY	0x02D2D2D2L
COLOR_DARK_SALMON	0x02FF4040L
COLOR_DARK_PEACH	0x02FF421EL
COLOR_GOLD	0x02FFBF18L
COLOR_YELLOW_GREEN	0x02E1E100L
COLOR_LIGHT_GREEN	0x0200E100L
COLOR_CARIBBEAN	0x0200E1ADL
COLOR_DK_PASTEL_BLUE	0x0200E0E0L
COLOR_DARK_CERULEAN	0x020082BFL
COLOR_MANGANESE_BLUE	0x020080FFL
COLOR_LILAC	0x024181FFL
COLOR_PURPLE	0x028242FFL
COLOR_LT_RED_VIOLET	0x02C140FFL
COLOR_LIGHT_MAGENTA	0x02FF42F9L
COLOR_ROSE	0x02FF40A0L
COLOR_CARNATION_PINK	0x02FF4070L
COLOR_25_GRAY	0x02C0C0C0L
COLOR_WATERMELON	0x02FF1F35L
COLOR_TANGERINE	0x02FF1F10L
COLOR_ORANGE	0x02FF8100L
COLOR_CHARTREUSE	0x02BFBF00L
COLOR_GREEN	0x0200C200L
COLOR_TEAL	0x0200C196L
COLOR_DARK_TURQUOISE	0x0200C1C2L
COLOR_LT_SLATE_BLUE	0x024181C0L
COLOR_MEDIUM_BLUE	0x020062E1L
COLOR_DARK_LILAC	0x024141FFL
COLOR_ROYAL_PURPLE	0x024200FFL
COLOR_FUCHSIA	0x02C200FFL
COLOR_CONFETTI_PINK	0x02FF22FFL
COLOR_PALE_BURGANDY	0x02F52B97L
COLOR_STRAWBERRY	0x02FF2259L
COLOR_30_GRAY	0x02B2B2B2L
COLOR_ROUGE	0x02E01F25L
COLOR_BURNT_ORANGE	0x02E12000L
COLOR_DARK_ORANGE	0x02E26200L
COLOR_LIGHT_OLIVE	0x02A1A100L
COLOR_KELLY_GREEN	0x0200A000L

COLOR_SEA_GREEN	0x02009F82L
COLOR_AZTEC_BLUE	0x02008080L
COLOR_DUSTY_BLUE	0x020060A0L
COLOR_BLUEBERRY	0x020041C2L
COLOR_VIOLET	0x020021BFL
COLOR_DEEP_PURPLE	0x024100C2L
COLOR_RED_VIOLET	0x028100FFL
COLOR_HOT_PINK	0x02FF00FFL
COLOR_DARK_ROSE	0x02FF0080L
COLOR_POPPY_RED	0x02FF0041L
COLOR_36_GRAY	0x02A2A2A2L
COLOR_CRIMSON	0x02C20000L
COLOR_RED	0x02FF0000L
COLOR_LIGHT_BROWN	0x02BF4100L
COLOR_OLIVE	0x02808000L
COLOR_DARK_GREEN	0x02008000L
COLOR_DARK_TEAL	0x02008250L
COLOR_SPRUCE	0x02006062L
COLOR_SLATE_BLUE	0x02004080L
COLOR_NAVY_BLUE	0x02001FE2L
COLOR_BLUE_VIOLET	0x024040C2L
COLOR_AMETHYST	0x024000A2L
COLOR_DK_RED_VIOLET	0x026000A1L
COLOR_MAGENTA	0x02E000E0L
COLOR_LIGHT_BURGUNDY	0x02DF007FL
COLOR_CHERRY_RED	0x02C20041L
COLOR_44_GRAY	0x028F8F8FL
COLOR_DARK_CRIMSON	0x02A00000L
COLOR_DARK_RED	0x02E10000L
COLOR_HAZELNUT	0x02A13F00L
COLOR_DARK_OLIVE	0x02626200L
COLOR_EMERALD	0x02006000L
COLOR_MALACHITE	0x0200603CL
COLOR_DARK_SPRUCE	0x02004041L
COLOR_STEEL_BLUE	0x02002F80L
COLOR_BLUE	0x020000FFL
COLOR_IRIS	0x022020A0L
COLOR_GRAPE	0x022200A1L
COLOR_PLUM	0x02400080L
COLOR_DARK_MAGENTA	0x02A1009FL
COLOR_BURGANDY	0x02C0007FL
COLOR_CRANBERRY	0x029F000FL
COLOR_50_GRAY	0x02808080L
COLOR_MAHOGANY	0x02600000L
COLOR_BRICK	0x02C21212L

COLOR_DARK_BROWN	0x02824200L
COLOR_DEEP_OLIVE	0x02424200L
COLOR_DARK_EMERALD	0x02004200L
COLOR_EVERGREEN	0x02004023L
COLOR_BALTIC_BLUE	0x0200323FL
COLOR_BLUE_DENIM	0x02002060L
COLOR_COBALT_BLUE	0x020020C2L
COLOR_DARK_IRIS	0x022222C0L
COLOR_MIDNIGHT	0x02000080L
COLOR_DARK_PLUM	0x021F007FL
COLOR_PLUM_RED	0x02800080L
COLOR_DARK_BURGANDY	0x02820040L
COLOR_SCARLET	0x02800000L
COLOR_63_GRAY	0x025F5F5FL
COLOR_CHESTNUT	0x02400000L
COLOR_TERRA_COTTA	0x02A11F12L
COLOR_UMBER	0x02604200L
COLOR_AMAZON	0x02212100L
COLOR_PEACOCK_GREEN	0x02002100L
COLOR_PINE	0x0200201FL
COLOR_SEAL_BLUE	0x02002041L
COLOR_DK_SLATE_BLUE	0x0200204FL
COLOR_ROYAL_BLUE	0x020000E0L
COLOR_LAPIS	0x020000A1L
COLOR_DARK_GRAPE	0x02000061L
COLOR_AUBERGINE	0x021F0062L
COLOR_DARK_PLUM_RED	0x0240005FL
COLOR_RASPBERRY	0x02620042L
COLOR_DEEP_SCARLET	0x02620012L
COLOR_69_GRAY	0x024F4F4FL
COLOR_BURNT_SIENNA	0x02200000L
COLOR_MILK_CHOCOLATE	0x02622100L
COLOR_BURNT_UMBER	0x02412000L
COLOR_DEEP_AVOCADO	0x02101000L
COLOR_DEEP_FOREST	0x02001000L
COLOR_DARK_PINE	0x0200120CL
COLOR_DK_METALIC_BLUE	0x0200121FL
COLOR_AIR_FORCE_BLUE	0x02001040L
COLOR_ULTRAMARINE	0x020000C2L
COLOR_PRUSSIAN_BLUE	0x020000AFL
COLOR_RAISIN	0x0200004FL
COLOR_EGGPLANT	0x02000040L
COLOR_BOYSENBERRY	0x02200042L
COLOR_BORDEAUX	0x02400040L
COLOR_RUBY	0x02410012L

COLOR_75_GRAY	0x02404040L
COLOR_RED_GRAY	0x02D0B1A1L
COLOR_TAN	0x02E0A175L
COLOR_KHAKI	0x02D2B06AL
COLOR_PUTTY	0x02C0C27CL
COLOR_BAMBOO_GREEN	0x0282C168L
COLOR_GREEN_GRAY	0x0281C097L
COLOR_BALTIC_GRAY	0x027FC2BCL
COLOR_BLUE_GRAY	0x0271B2CFL
COLOR_RAIN_CLOUD	0x02B1B1D2L
COLOR_LILAC_GRAY	0x029F9FE0L
COLOR_LT_PURPLE_GRAY	0x02C0A1E0L
COLOR_LIGHT_MAUVE	0x02E29FDEL
COLOR_LT_PLUM_GRAY	0x02EF91EBL
COLOR_LT_BURGANDY_GRAY	0x02E29FC8L
COLOR_ROSE_GRAY	0x02F18FBCL
COLOR_82_GRAY	0x022F2F2FL
COLOR_DARK_RED_GRAY	0x027F604FL
COLOR_DARK_TAN	0x02A16252L
COLOR_SAFARI	0x02806210L
COLOR_OLIVE_GRAY	0x0282823FL
COLOR_JADE	0x023F621FL
COLOR_DK_GREEN_GRAY	0x023C613EL
COLOR_SPRUCE_GRAY	0x0237605EL
COLOR_DK_BLUE_GRAY	0x02104160L
COLOR_ATLANTIC_GRAY	0x02424282L
COLOR_DK_LILAC_GRAY	0x026260A1L
COLOR_PURPLE_GRAY	0x02624181L
COLOR_MAUVE	0x02603181L
COLOR_PLUM_GRAY	0x02602162L
COLOR_BURGUNDY_GRAY	0x02622152L
COLOR_DARK_ROSE_GRAY	0x02813F62L
COLOR_BLACK	0x02000000L
COLOR_TRANSPARENT	0xFFFFFFFFL
	0xFFFFFFFFL

Database connection error codes

DBErr_Already_Connected
DBErr_Bad_Database_ID
DBErr_ConnectToFailed
DBErr_ConnListField
DBErr_ConnListTable
DBErr_Execute_Open
DBErr_Execute_Query
DBErr_GetValue_Convert

DBErr_Invalid_Column
DBErr_InvalidParam
DBErr_NoConnection
DBErr_NoConnSet
DBErr_NoDataSource
DBErr_NoDataSources
DBErr_NoQuery
DBErr_NoRows
DBErr_NotConnected
DBErr_Option_Unsupported
DBErr_Picture
DBErr_ReadMemo
DBErr_SetConvert
DBErr_Update
DBErr_Optimistic
DBErr_Override
DBErr_Pessimistic

Approach LotusScript Enumeration Values

Approach scripts use the following enumeration values to specify the various types of display objects available in Approach.

Note When you use an enumeration value in a script, you MUST prepend a dollar sign ($) to the enumeration to distinguish the enumerator from a variable.

Object positions	LtsPositionTopLeft
	LtsPositionTop
	LtsPositionTopRight
	LtsPositionLeft
	LtsPositionCenter
	LtsPositionRight
	LtsPositionBottomLeft
	LtsPositionBottom
	LtsPositionBottomRight
	LtsPositionNone
Border relief styles	LtsReliefNone
	LtsReliefRaised
	LtsReliefLowered
Picture Display options	LtsPictureDisplayCrop
	LtsPictureDisplayShrink
Line object orientation	LtsOrientationNegSlope
	LtsOrientationPosSlope
Line styles	LtsLineStyleNone
	LtsLineStyleSolid
	LtsLineStyleDot
	LtsLineStyleShortDash
	LtsLineStyleMediumDash
	LtsLineStyleLineLongDash
	LtsLineStyleMediumShortDash
	LtsLineStyleMediumShortShortDash
	LtsLineStyleDashDot
	LtsLineStyleDashDotDot
	LtsLineStyleDouble
Line spacing	LtsLineSpacingSingle
	LtsLineSpacingSingleAndHalf
	LtsLineSpacingDouble
Display object alignment	LtsAlignmentLeft
	LtsAlignmentRight
	LtsAlignmentHorizCenter
	LtsAlignmentSmart
	LtsAlignmentJustify
	LtsAlignmentTop
	LtsAlignmentVertCenter
	LtsAlignmentBottom
Line widths	AprHairline

	AprHalfPoint
	Apr1Point
	Apr2Point
	Apr3Point
	Apr6Point
	Apr12Point
Border patterns	LtsBorderPatternNone
	LtsBorderPatternSolid
	LtsBorderPatternDashDot
	LtsBorderPatternDashDotDot
	LtsBorderPatternLongDash
	LtsBorderPatternDashed
	LtsBorderPatternDot
	LtsBorderPatternRaised
	LtsBorderPatternLowered
	LtsBorderPatternDouble
	LtsBorderStyleNone
View object types	AprPicture
	AprPicturePlus
	AprButton
	AprLineObject
	AprEllipse
	AprRectangle
	AprRoundRect
	AprOleObject
	AprFieldBox
	AprListBox
	AprCheckBox
	AprDropDownBox
	AprRadioButton
	AprTextBox
View types	AprForm
	AprReport
	AprChart
	AprCrosstab
	AprWorksheet
	AprEnvelope
	AprMailingLabel
	AprFormLetter
Panel types	AprBodyPanel
	AprHeaderFooter
	AprRepeatingPanel
	AprSummaryPanel
	AprSummaryTrailing
	AprSummaryLeading

	AprListField
	AprListUser
Field types	AprFieldNumber
	AprFieldText
	AprFieldPicture
	AprFieldDate
	AprFieldTime
	AprFieldBool
	AprFieldMemo
	AprFieldCalculation
	AprFieldVariable
Summary calculations	AprCalcAverage
	AprCalcCount
	AprCalcSum
	AprCalcMin
	AprCalcMax
	AprCalcStDev
	AprCalcVariance
Chart types	AprChartTypeBar
	AprChartTypeStackBar
	AprChartType100PStackBar
	AprChartTypeHorizBar
	AprChartTypeHorizStackBar
	AprChartType25DBar
	AprChartType25DStackBar
	AprChartType100P25DStackBar
	AprChartType25DHorizBar
	AprChartType25DHorizStackBar
	AprChartTypeLine
	AprChartTypeArea
	AprChartTypeMixed
	AprChartTypePie
	AprChartTypeMultiplePie
	AprChartType25DLine
	AprChartType25DArea
	AprChartType25DMixed
	AprChartType3DPie
	AprChartType3DMultiplePie
	AprChartType3DBar
	AprChartType3DLine
	AprChartType3DArea
	AprChartType3DMixed
	AprChartTypeHLCO
	AprChartTypeRadar
	AprChartTypeXY

AprChartTypeTable
AprChartTypeGANTT
AprChartTypeOrg
AprChartType25DHorBar
AprChartType25DHorStackBar
AprChartTypeSurface
AprChartType3DStackBar
AprChartType100P3DStackBar
AprChartType100PHorizBar
AprChartType100P25DHorizBar
AprChartTypeHLCOJapan
AprChartTypeRadarArea

Approach LotusScript Events A-Z

A

(None)

B

Broadcast event

C

CellDataChange event
CellGetFocus event
CellLostFocus event
Change event
Click event
Closewindow event

D

DocumentClose event
DocumentCreated event

U
UserTimer event

V
ViewSwitch event

W
(None)

X
(None)

Y
(None)

Z
(None)

Approach LotusScript Methods A-Z

A

- Activate method
- Add method
- Add method (Collection class)
- Add method (Sort class)
- AddColumn method
- AddRow method
- AddListItem method
- And method

B

- BringToFront method
- Browse method

C

- Cascade method
- Close method

D

E

F

G

H

HideRecord method

I

InsertAfter method
IsCommandChecked method
IsCommandEnabled method
IsEmpty method

J

(None)

K

(None)

L

LastRecord method
LastRow method
ListDataSources method
ListFields method
ListTables method

M

MakeNamedStyle method
Maximize method
Merge method
Minimize method

N

New method (Button class)
New method (ChartView class)
New method (CheckBox class)
New method (Collection class)
New method (Color class)
New method (Connection class)
New method (Crosstab class)
New method (Document class)
New method (DropdownBox class)
New method (Ellipse class)
New method (FieldBox class)
New method (Find class)
New method (FindDistinct class)
New method (FindDuplicate class)
New method (FindTopLowest class)
New method (Form class)
New method (LineObject class)
New method (ListBox class)
New method (Picture class)
New method (PicturePlus class)
New method (Query class)
New method (RadioButton class)
New method (Rectangle class)
New method (RepeatingPanel class)
New method (Report class)
New method (ResultSet class)
New method (RoundRect class)
New method (Sort class)
New method (SummaryPanel class)
New method (TextBox class)

U

Unsort method
UpdateRow method

V

(None)

W

(None)

X

(None)

Y

(None)

Z

Zoom method

Approach LotusScript Properties A-Z

A

B

I

IconBarVisible property
IsBeginOfData property
IsChecked property
IsClicked property
IsConnected property
IsEndOfData property
IsReadOnly property
IsResultSetAvailable property
Italic property

J

(None)

K

KeepRecsTogether property
Keywords property

L

LabelAlignment property
LabelFont property
LabelPosition property
LabelText property
Language property
LastModified property
Left property
Left property (Border class)
LineSpacing property
LineStyle property
Location property
LPObject property

M

MacroClick property
MacroDataChange property
MacroTabIn property
MacroTabOut property
Magenta property
MainTable property
MenuBar property
Menus property
Modified property

N

Name property
NamedFindSort property
NamedFindSorts property
NamedStyle property
NamedStyles property
NonPrinting property
NumColumns property
NumFields property
NumJoins property
NumLines property
NumPages property
NumRecords property
NumRecordsFound property
NumRevisions property
NumTables property

Approach: ApplicationWindow class

The ApplicationWindow object describes the main Approach window for a running application. There is one application window for each running Approach executable.

Contained by

Each instance of an ApplicationWindow object is identified from its parent through an expanded property. The following classes can contain an ApplicationWindow object:

Class
Application

Usage

ApplicationWindow class is at the top of the window containment hierarchy for the application.

Use an ApplicationWindow to control menu and application window commands, such as the following:

- Perform menu commands
- Minimize, maximize, and close the application window

ApplicationWindow class members

Properties

Application AS Application class
Parent AS Application class

Methods

Cascade
Close
DoMenuCommand
GetHandle
IsCommandChecked
IsCommandEnabled
Maximize
Minimize
Restore
Tile

Events

None

Approach: Application class

The Application object maintains application-wide settings and user information for an Approach session. There is only one application object per running application, that is, one application per Approach executable running. There is a single application window associated with each application. The application window can contain multiple document windows (.APR files).

Contained by

Class
None

Usage

Use the Application class to find information about the current Approach session, such as the following:

- Which document (.APR file) is active
- The application's path
- The active view

Application class members

Properties

ActiveDocument AS Document class
AS DocWindow class
ActiveView AS View class
Application AS Application class
ApplicationWindow AS ApplicationWindow class
Documents AS BaseCollection class
FullName
Language
Name
Parent AS Application class
Path
Selection
Visible
Windows AS BaseCollection class

Methods

GetColorFromRGB
OpenDocument
Quit
RunProcedure

Events

Broadcast
DocumentClose
DocumentCreated
DocumentOpened
MailCheck
MailSend
Quit

Approach: Background class

The background style for display elements that have a background and use the Background property. You can modify the color of the background.

Contained by

Each instance of a Background object is identified from its parent through an expanded property. The following classes can contain a Background object:

Class
CheckBox
DropDownBox
Ellipse
FieldBox
ListBox
Panel
RadioButton
Rectangle
RoundRect
TextBox

Usage

This class modifies the backgrounds of display elements that have the Background property. You cannot display a background object; it is used as the Background property of another object.

Background class members

Properties

Color AS Color class

Methods

None

Events

None

Approach: BaseCollection class

BaseCollection determines the number of objects in a collection and whether or not the collection is empty.

Contained by

Class
None

Usage

The BaseCollection class is an abstract class, so you cannot create an instance of BaseCollection.

You can, however, represent all its derived classes with the BaseCollection class. For example, you can write a procedure that takes "Source As BaseCollection as a parameter. This allows you to use any instance of a derived BaseCollection class in that procedure.

You can also use LotusScript functions to cycle through a Collection class.

BaseCollection class members

Properties

Count

Methods

IsEmpty

Events

None

Approach: BodyPanel class

The area of a form on which to display data from individual records.

Contained by

Each instance of a BodyPanel object is identified from its parent through an expanded property. The following classes can contain a BodyPanel object:

Class
ChartView
Crosstab
Envelope
Form
FormLetter
MailingLabels
Report
Worksheet

Usage

Configure some important characteristics of body panel using the following properties:

- Set the color of the header and footer panels using the Background, and Border properties
- Determine the printing behavior, such as the expansion or reduction of the size of the panel to accommodate the data, using the Expand and Reduce properties

BodyPanel class members

Properties

Background AS Background class
Border AS Border class
Expand
Height
NamedStyle
Parent AS View class
Reduce
Type

Methods

MakeNamedStyle

Events

None

Approach: Border class

The border style for displayable elements.

Contained by

Each instance of a Border object is identified from its parent through an expanded property. The following classes can contain a Border object:

Class
BodyPanel
DropDownBox
Ellipse
FieldBox
HeaderFooterPanel
ListBox
Rectangle
RepeatingPanel
RoundRect
SummaryPanel
TextBox

Usage

Create and modify border objects. Once you have a border object, you can use it as the property of any object with a border property.

You can also specify border characteristics by defining a named style and associating the named style with a display element object.

Approach: Border class members

Properties

- BaseLine
- Bottom
- Color AS Color class
- EncloseLabel
- Left
- Pattern
- Right
- Top
- Width

Methods

- None

Events

- None

Approach: Button class

A display element that, when users click it, can run an attached macro or script.

Contained by

Each instance of an Button object is identified from its parent through an expanded property. The following classes can contain a Button object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Buttons with attached macros or scripts can help ensure users' successful use of your application by focusing their activity on choices you set up for them, responding to their input, and guiding them on predetermined paths through the application. For example, create a form with buttons arranged as a menu of choices. When users click a button, the application responds by switching to another view designed to complete the task described by the menu item.

The coordinate system for display elements is measured in twips.

The coordinate system is measured from the top left corner of the panel in which you place the element. For example, a button placed in the body panel of a report is based on the top left corner of the body panel, not the report.

Configure some important characteristics of a button using the following properties:

- To set the macro to execute when a user clicks the button, use the MacroClick property.
- To make the button available to the user, use the Enabled property.

Approach: Button class members

Properties

- Enabled
- Font AS Font class
- Height
- Left
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Text
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: ChartView class

A ChartView object is an Approach chart.

Contained by

Each instance of a ChartView object is identified from its parent through an expanded property. The following classes can contain a ChartView object:

Class
Document

Usage

Configure some important characteristics of a chart using the following properties:

- Determine the document containing the chart using the Document property.
- Determine the main table used in the chart using the MainTable property.
- Set the macros you want to run when users switch to or from the view using the OnSwitchFromMacro and OnSwitchToMacro properties.

ChartView class members

Properties

Document AS Document class
MainTable
MenuBar
Name
OnSwitchFromMacro
OnSwitchToMacro
Parent AS Document class
TimerInterval
Type
Visible

Methods

New

Events

SwitchFrom
SwitchTo
UserTimer

Approach: CheckBox class

A display element indicating a yes/no or a true/false condition. A check box can be checked or unchecked.

Contained by

Each instance of a CheckBox object is identified from its parent through an expanded property. The following classes can contain a CheckBox object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

A check box can serve any one of the following functions:

- Display a field value, allowing users to modify the value.
- Display a field value, without allowing users to modify the value (read-only).
- Provide a user-interface control whose true/false value is not connected to a database field.

Configure some important characteristics of a check box using the following properties and methods:

- Set the values written to a field for each check box state using the CheckedValue and UncheckedValue properties.
- Set a default value for a new check box using the SetState method.
- Set the label for the check box using the LabelText property.

CheckBox class members

Properties

- Background AS Background class
- CheckedValue
- DataField
- DataTable
- Height
- IsChecked
- LabelAlignment
- LabelFont AS Font class
- LabelPosition
- LabelText
- Left
- MacroDataChange
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ReadOnly
- Relief
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- UncheckedValue
- Value
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus
- SetState

Events

- Change
- Click
- DoubleClick
- GotFocus
- LostFocus
- MouseDown
- MouseMove
- MouseUp

Approach: Collection class

The Collection class defines a list of objects and allows you to add, remove, and rearrange the objects in the list.

Contained by

Class
None

Usage

Use a Collection object to add, remove, rearrange, or merge objects in a form, report, form letter, mailing label, or envelope. The Collection object is available through the ObjectList property of any of these views.

For example, to combine the objects that appear on Form1 and Form2 in the same document (.APR file), use the Merge method of the Collection object.

In this script, the ObjectList property of the Form1 and Form2 objects identifies the Collection objects:

```
Dim TotalNumObjects As Integer

TotalNumObjects =
CurrentDocument.Form1.ObjectList.Merge(CurrentDocument.Form2.ObjectList)
```

Collection class members

Properties

Count

Methods

Add
IsEmpty
Merge
New
Remove
SetAt

Events

None

Approach: Color class

The Color class defines a color for objects that have Color as a property.

Contained by

Each instance of a Color object is identified from its parent through an expanded property. The following classes can contain a Color object:

Class
Background
Border
Font
LineStyle

Usage

Determine the current levels of red, green, and blue (RGB) that contribute to a specific color.

You cannot change the individual RGB components of an existing color, you have to create a new color object and base its RGB settings on the original color.

A color object cannot be displayed; it can only be the Color property of another object.

Color class members

Properties

- Black
- Blue
- Cyan
- Green
- Magenta
- Red
- Transparent
- Yellow

Methods

- GetRGB
- New
- SameColor
- SetRGB

Events

- None

Approach: Connection class

Connecting to a database server.

Contained by

Each instance of a Connection object is identified from its parent through an expanded property. The following classes can contain a Connection object:

Class
Query

Usage

The Connection class is one of three classes that together allow you to access data in a batch process, bypassing the Approach user interface. Accessing data through this batch process can be very fast for the following operations:

- Examining or modifying many or all records in a very large database
- Performing calculations using data from a large number of records

Creating a Connection object requires the following operations:

- Declaring and initializing a variable as type Connection using the New method
- Executing the connection using the ConnectTo method
- If needed, specifying how Approach commits data changes to the table using the AutoCommit property and Transactions method

You can use the same Connection object to access more than one table of the same data source type.

If you want to access data in a table that is already associated with your application, use the CreateResultSet method from the Table class.

Connection class members

Properties

- AutoCommit
- DataSourceName
- IsConnected
- Password
- UserID

Methods

- ConnectTo
- Disconnect
- GetError
- GetErrorMessage
- GetExtendedErrorMessage
- ListDataSources
- ListFields
- ListTables
- New
- Transactions

Events

- None

Approach: Crosstab class

Crosstab object is an Approach crosstab.

Contained by

Each instance of a Crosstab object is identified from its parent through an expanded property. The following classes can contain a Crosstab object:

Class
Document

Usage

Configure some important characteristics of a crosstab using the following properties:

- Determine the document that contains the crosstab using the Document property.
- Apply the current found set to the crosstab using the ApplyFoundSet property.
- Set the drill-down view for the crosstab using the DrillDownView property.

Crosstab class members

Properties

- ApplyFoundSet
- Document AS Document class
- DrillDownView AS View class
- MainTable
- MenuBar
- Name
- OnSwitchFromMacro
- OnSwitchToMacro
- Parent AS Document class
- PrintDate
- PrintPageNum
- PrintTitle
- ShowRelated
- TimerInterval
- Title
- Type
- Visible

Methods

- New

Events

- SwitchFrom
- SwitchTo
- UserTimer

Approach: Display class

The base class of the derived classes you work with to construct a user interface, for example, buttons to start scripts; controls like check boxes and list boxes; geometric shapes; and text blocks.

The following Approach classes inherit the members of the Display class:

Button	LineObject	RadioButton
CheckBox	ListBox	Rectangle
DropdownBox	OLEObject	RoundRect
Ellipse	Picture	TextBox
FieldBox	PicturePlus	

Contained by

Each instance of a Display object is identified from its parent through an expanded property. The following classes can contain a Display object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

The Display class is an abstract class, so you cannot create an instance of Display.

You can, however, represent all its derived classes with the Display class. For example, you can write a procedure that takes "Source As Display" as a parameter. This allows you to pass any instance of a derived Display class to that procedure.

The coordinate system for display elements is measured in twips.

The coordinate system is based on the top left corner of the panel in which you place the element. For example, if you have a picture in the header panel of a report, the picture's coordinates are based on the top left corner of the header panel, not the report. Similarly, a check box placed in the body panel of the report is based on the top left corner of the body panel, not the report.

Approach: Display class members

Properties

- Height
- Left
- MacroTabIn
- MacroTabOut
- Name
- NonPrinting
- Page
- Parent AS View class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- Refresh
- SendToBack
- SetFocus

Events

None

Approach: Document class

A Document object describes the basic top-level document (.APR file) for an Approach application.

Contained by

Each instance of a Document object is identified from its parent through an expanded property. The following classes can contain a Document object:

Class
Application

Usage

A document contains information such as the following:

- File name and path of the document.
- The owner or author's name
- A description of the document
- Creation and modification dates of the document
- The databases associated with the document
- The macros and named find/sorts associated with the document
- Calculated fields in the document

Document class members

Properties

Author
CalcTable AS Table class
CreateDate
Description
FileName
FullName
Keywords
LastModified
Menus
Modified
Name
NamedFindSorts
NamedStyles
NumJoins
NumRevisions
NumTables
NumViews
Parent AS Application class
Path
Tables AS BaseCollection class
User
VarTable AS Table class
Views AS BaseCollection class
Window AS DocWindow class

Methods

Activate
CreateCalcField
DeleteCalcField
GetTableByName
New

Events

None

Approach: DocWindow class

A DocWindow object provides access to the current state of the document, including such information as the current record and view, and the sets of icons or buttons that are available.

Contained by

Each instance of a DocWindow object is identified from its parent through an expanded property. The following classes can contain a DocWindow object:

Class
Document

Usage

A DocWindow object is a view in which to perform database operations, such as the following tasks:

- Navigate between fields in a view.
- Navigate between records in the current found set.
- Enter data into fields.

DocWindow class members

Properties

ActionBarVisible
ActiveView AS View class
CurrentFind AS Find class
CurrentRecord
CurrentSort AS Sort class
Document AS Document class
IconBarVisible
NamedFindSort
NumRecordsFound
Parent AS Document class
Redraw
StatusBarVisible
ViewTabVisible
Visible

Methods

Browse
Close
Copy
CopyView
CountRecords
Cut
DeleteFile
DeleteFoundSet
DeleteRecord
DuplicateRecord
FillField
FindAll
FindSort
FirstRecord
GetHandle
GoToRecord
HideRecord
LastRecord
Maximize
Minimize
NewRecord
NextRecord
Paste
PrevRecord
Print
PrintPreview
Refresh
Repaint
Replicate
Restore
SaveChanges
SaveViewAsJDoc
SelectAll
TabNext
TabPrev
TabTo
Unsort
Zoom

Events

CloseWindow
NewRecord
OpenWindow
RecordChange
RecordCommit
ViewSwitch

Approach: DropDownBox class

A display element that displays a list of values in a drop-down box. Users can select only one item in the list.

Contained by

Each instance of a DropDownBox object is identified from its parent through an expanded property. The following classes can contain a DropDownBox object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

A drop-down box displays a list of selections for a user. For example, a drop-down box can display the existing values in a field to simplify data entry.

Configure some important characteristics of a drop-down box using the following properties and methods:

- Set the drop-down box so users cannot enter data that does not already exist in the list using the Editable property.
- Set the list of items you want to display in the drop-down box using the SetFieldList or SetList methods.

DropDownBox members

Properties

- Alignment
- Background AS Background class
- Border AS Border class
- DataField
- DataTable
- Editable
- Font AS Font class
- Height
- LabelAlignment
- LabelFont AS Font class
- LabelPosition
- LabelText
- Left
- MacroDataChange
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ReadOnly
- ShadowColor AS Color class
- ShowArrow
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Text
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFieldList
- SetFocus
- SetList

Events

- Change
- GotFocus
- LostFocus

Approach: Ellipse class

A display element that you can add to a form, report, mailing label, form letter, envelope, or chart.

An ellipse with its height equal to its width is a circle.

Contained by

Each instance of an Ellipse object is identified from its parent through an expanded property. The following classes can contain an Ellipse object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Ellipses can enhance views of data. For example, use an ellipse to draw attention to instructions in a text block.

Configure some important characteristics of an ellipse display element using the following properties and methods:

- Create a circle by setting the Height and Width properties of the ellipse display elements to the same measurements.
- Set the background and shadow color of the ellipse display elements using the Background and ShadowColor properties.
- Set the position of the ellipse display elements in a group of overlapping objects by using the BringToFront and SendToBack methods.

Ellipse class members

Properties

- Background AS Background class
- Border AS Border class
- Height
- Left
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: Envelope class

An Envelope object is an Approach envelope.

Contained by

Each instance of an Envelope object is identified from its parent through an expanded property. The following classes can contain an Envelope object:

Class
Document

Usage

Configure some important characteristics of an envelope using the following properties:

- Determine the document that contains the envelope using the Document property.
- Set whether to display margins in the envelope view using the HideMargins property.
- Determine the display elements that appear in the view using the ObjectList property.

Envelope class members

Properties

Document AS Document class
HideMargins
MainTable
MenuBar
Name
ObjectList AS Collection class
ObjectListAll AS Collection class
OnSwitchFromMacro
OnSwitchToMacro
Parent AS Document class
TimerInterval
Type
Visible

Methods

None

Events

SwitchFrom
SwitchTo
UserTimer

Approach: FieldBox class

A display element that displays data and allows users to enter data. A field box may or may not be bound to a field in a table.

Contained by

Each instance of a FieldBox object is identified from its parent through an expanded property. The following classes can contain a FieldBox object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

A field box is the basic data-entry type for viewing and entering data.

- A field box object is **bound** when you define it to display the data from a table field by assigning it to the DataField property and if necessary, the DataTable property.
- A field box is **unbound** when it is not associated with a table field. You might use an "unbound" field box to provide a value for a calculation. For example, a report might use a field box in which users enter a value that is then used to determine which records are summarized in the report.

FieldBox class members

Properties

Alignment
Background AS Background class
Border AS Border class
DataField
DataTable
Expand
Font AS Font class
Height
LabelAlignment
LabelFont AS Font class
LabelPosition
LabelText
Left
MacroDataChange
MacroTabIn
MacroTabOut
Name
NamedStyle
NonPrinting
Page
Parent AS View class
ReadOnly
Reduce
ShadowColor AS Color class
ShowInPreview
SlideLeft
SlideUp
TabOrder
TabStop
Text
Top
Type
Visible
Width

Methods

BringToFront
InsertAfter
MakeNamedStyle
New
Refresh
SendToBack
SetFocus

Events

Change
Click
DoubleClick
GotFocus
KeyDown
KeyPress
KeyUp
LostFocus
MouseDown
MouseMove
MouseUp

Approach: FindDistinct class

The conditions used to find records with distinct values.

Contained by

Class	Property
Find	FindSpecial

Usage

A FindDistinct object consists of one or more find conditions that indicate in which fields to search for distinct values. The found set includes one record with each value Approach finds in the field or fields specified.

Defining and running a find in LotusScript involves the following operations:

- Creating a FindDistinct object using the New method
- Adding conditions to the find using the Add method
- (Optional) Adding the FindDistinct object to an existing Find object using the FindSpecial property
- Executing the find using the DocWindow object FindSort method

If a found set exists when you run a FindDistinct operation, Approach finds the distinct records in the found set.

FindDistinct class members

Properties

None

Methods

Add
GetAt
GetCount
New

Events

None

Approach: FindDuplicate class

The conditions used to find records with duplicate values.

Contained by

Class	Property
Find	FindSpecial

Usage

A FindDuplicate object consists of one or more find conditions that indicate in which fields to search for duplicate values. When you create a FindDuplicate object, you define one of these conditions.

Defining and running a find in LotusScript involves the following operations:

- Creating a FindDuplicate object using the New method
- Adding other conditions to the find using the Add method
- Indicating whether to include all records in the found set or leave one of each duplicated set out of the found set using the ExcludeFirst property
- (Optional) Adding the FindDuplicate object to an existing Find object using the FindSpecial property
- Executing the find using the DocWindow object FindSort method

If a found set exists when you run a FindDuplicate operation, Approach finds the duplicate records in the found set.

FindDuplicate class members

Properties

ExcludeFirst

Methods

Add
GetAt
GetCount
New

Events

None

Approach: FindTopLowest class

The conditions used to find records with the top or lowest values in the table.

Contained by

Class	Property
Find	FindSpecial

Usage

A FindTopLowest object consists of one find conditions that indicates in which field to search for the top or lowest number or percent of values.

Defining and running a find in LotusScript involves the following operations:

- Creating a FindTopLowest object using the New method
- (Optional) Adding the FindTopLowest object to an existing Find object using the FindSpecial property
- Executing the find using the DocWindow object FindSort method

If a found set exists when you run a FindTopLowest operation, Approach finds the top or lowest records in the found set.

FindTopLowest class members

Properties

None

Methods

GetAt
New

Events

None

Approach: Find class

The conditions used to find records.

Contained by

Class
None

Usage

A Find object consists of one or more find conditions that indicate which fields and values to search for. When you create a Find object, you define one of these conditions.

Defining and running a find in LotusScript involves the following operations:

- Creating a Find object using the New method
- Adding conditions to the find using the And and Or methods and FindSpecial property
- Executing the find using the DocWindow object FindSort method

When you build find conditions, the order the conditions are added to the find is important. Approach evaluates each condition as it appears, independent of the conditions which follow.

For example, to find records in an Orders database that correspond to sales over $1000 in either Japan or the US, build the Find object as follows:

```
Dim SalesFind As New Find

Call SalesFind.And (OrderTotal, "> 1000")

Call SalesFind.And (Country, "Japan")

Call SalesFind.Or (OrderTotal, "> 1000")

Call SalesFind.And (Country, "USA")
```

The OrderTotal condition must be repeated to find the proper set of records.

If you are unsure about how Approach will evaluate find conditions, build the find in a view while you are recording the steps. Choose Edit - Record Transcript.

For more information, see the FindDistinct, FindDuplicate, FindTopLowest, and Sort classes.

Find class members

Properties

FindSpecial

Methods

And
GetAt
GetCount
New
Or

Events

None

Approach: Font class

A text style for display elements that have text and use the Font property. You can modify the color, size, and name of the font.

Contained by

Each instance of a Font object is identified from its parent through an expanded property. The following classes can contain a Font object:

Class
Button
DropDownBox
FieldBox
ListBox
TextBox

Usage

Use this class to modify text of display elements that have the Font property. A font object cannot be displayed; it is used as the Font property of another object.

Font class members

Properties

- Bold
- Color AS Color class
- FontName
- Italic
- Relief
- Size
- Strikethrough
- Underline

Methods

- None

Events

- None

Approach: FormLetter class

A FormLetter object is an Approach form letter.

Contained by

Each instance of a FormLetter object is identified from its parent through an expanded property. The following classes can contain a FormLetter object:

Class
Document

Usage

Configure some important characteristics of a form letter using the following properties:

- Determine the document that contains the form letter using the Document property.
- Determine which object is currently selected using the Selection property.
- Determine the display elements that appear in the view using the ObjectList property.

FormLetter class members

Properties

CurrentPageNum
Document AS Document class
HideMargins
MainTable
MenuBar
Name
ObjectList AS Collection class
ObjectListAll AS Collection class
OnSwitchFromMacro
OnSwitchToMacro
Parent AS Document class
Selection
TimerInterval
Type
Visible

Methods

None

Events

SwitchFrom
SwitchTo
UserTimer

Approach: Form class

A view that can contain display elements and panels, and displays the data of one record at a time. It is typically used for entering data.

Contained by

Each instance of a Form object is identified from its parent through an expanded property. The following classes can contain a Form object:

Class
Document

Usage

A form gives access to one or more fields in a record. After you create a form, you can add panels and display elements such as field boxes, check boxes, rectangles, and lines, to the BodyPanel of the Form object. You can identify the BodyPanel using the expanded property Body. For example, the following dot notation identifies the field box fbxLastName on the form MyForm:

```
CurrentDocument.MyForm.Body.fbxLastName
```

Configure some important characteristics of a form view using the following properties and methods:

- Determine the document that contains the form using the Document property.
- Determine which page of the form users are viewing using the CurrentPageNum property.
- Change the number of pages in the view using the NewPage and DeletePage methods.
- Display the form as a dialog box using the ShowAsDialog property.

Form class members

Properties

- CurrentPageNum
- Document AS Document class
- HideMargins
- MainTable
- MenuBar
- Name
- NumPages
- ObjectList AS Collection class
- ObjectListAll AS Collection class
- OnSwitchFromMacro
- OnSwitchToMacro
- Parent AS Document class
- Selection
- TimerInterval
- Type
- Visible

Methods

- DeletePage
- New
- NewPage

Events

- PageSwitch
- SwitchFrom
- SwitchTo
- UserTimer

Approach: HeaderFooterPanel class

A header or footer for an Approach report.

Contained by

Each instance of a HeaderFooterPanel object is identified from its parent through an expanded property. The following classes can contain a HeaderFooterPanel object:

Class
Report

Usage

Configure some important characteristics of header and footer panels using the following properties:

- Set the color of the header and footer panels using the Background, and Border properties
- Determine the height (in twips) of the header or footer panel using the Height property.

A new report created in LotusScript contains two HeaderFooterPanel objects: a header panel and a footer panel. These objects have height zero until you change their size.

HeaderFooterPanel class members

Properties

Background AS Background class
Border AS Border class
Height
NamedStyle
Parent AS View class
Type

Methods

MakeNamedStyle

Events

None

Approach: LineObject class

A horizontal, vertical, or diagonal line on a form, report, mailing label, form letter, envelope, or chart.

Contained by

Each instance of a LineObject object is identified from its parent through an expanded property. The following classes can contain a LineObject object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Use a LineObject object to add lines to an Approach view. For example, use a LineObject to divide fields into groups.

Configure some important characteristics of a line using the following properties and methods:

- Determine the color and style of the line using the LineStyle property.
- Determine shadowcolor of the line using the ShadowColor property.
- Whether the line appears in Print Preview, or when you print the view containing the line using the Nonprinting and ShowInPreview properties.
- The position in the layers of the view in relation to other display elements using the BringToFront and SendToBack methods.

LineObject class members

Properties

Background AS Background class
Height
Left
LineStyle AS LineStyle class
MacroClick
MacroTabIn
MacroTabOut
Name
NamedStyle
NonPrinting
Orientation
Page
Parent AS View class
ShadowColor AS Color class
ShowInPreview
SlideLeft
SlideUp
TabOrder
TabStop
Top
Type
Visible
Width

Methods

BringToFront
InsertAfter
MakeNamedStyle
New
Refresh
SendToBack
SetFocus

Events

Click
DoubleClick
MouseDown
MouseMove
MouseUp

Approach: LineStyle class

A line style for objects that have LineStyle as a property.

Contained by

Each instance of a LineStyle object is identified from its parent through an expanded property. The following classes can contain a LineStyle object:

Class
LineObject

Usage

The LineStyle class creates and modifies line styles used for display objects. A LineStyle object cannot be displayed; it is used as the LineStyle property of another object.

LineStyle class members

Properties

Color AS Color class
Pattern
Width

Methods

None

Events

None

Approach: ListBox class

A display element that displays a list of values. A list box is a sizeable scrolling list.

Contained by

Each instance of a ListBox object is identified from its parent through an expanded property. The following classes can contain a ListBox object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

A list box can serve the following functions:

- Display a list of values that users cannot change.
- Display a field value, without allowing users to modify the value (read-only).
- Limit the values the user can enter in a database to the values you add to the list.

Configure some important characteristics of a list box using the following properties and methods:

- Set the values available in a list box using the AddListItem and RemoveListItem methods.
- Set a default value for a list box using the CurrentSelection property.
- Set the label for the list box using the LabelText property.

ListBox class members

Properties

- Background AS Background class
- Border AS Border class
- Count
- CurrentSelection
- DataField
- DataTable
- Font AS Font class
- Height
- LabelAlignment
- LabelFont AS Font class
- LabelPosition
- LabelText
- Left
- MacroDataChange
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ReadOnly
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Text
- Top
- Type
- Visible
- Width

Methods

- AddListItem
- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- RemoveListItem
- SendToBack
- SetFieldList
- SetFocus
- SetList

Events

- Change
- Click
- DoubleClick
- GotFocus
- LostFocus
- SelectionChange

Approach: MailingLabels class

A mailing labels object in an Approach view that contains mailing labels.

Contained by

Each instance of a MailingLabels object is identified from its parent through an expanded property. The following classes can contain a MailingLabels object:

Class
Document

Usage

Configure some important characteristics of mailing labels using the following properties:

- Determine the document that contains the mailing labels using the Document property.
- Set whether to display margins in the mailing label view using the HideMargins property.
- Determine the display elements that appear in the view using the ObjectList property.

MailingLabels class members

Properties

Document AS Document class
HideMargins
MainTable
MenuBar
Name
ObjectList AS Collection class
ObjectListAll AS Collection class
OnSwitchFromMacro
OnSwitchToMacro
Parent AS Document class
Selection
TimerInterval
Type
Visible

Methods

None

Events

SwitchFrom
SwitchTo
UserTimer

Approach: OLEObject class

Note OLEObject is not supported under OS/2.

An OLE object from any OLE-enabled application.

Contained by

Each instance of an OLEObject object is identified from its parent through an expanded property. The following classes can contain an OLEObject object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

OLE objects bring in objects from other applications, such as a document from a word processor or a spreadsheet from a spreadsheet program. An OLE object can be an entire file, such as a spreadsheet, or part of a file, such as a chart from a spreadsheet.

OLEObject class members

Properties

- Dispatch
- Height
- Left
- LPObject
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NonPrinting
- Page
- Parent AS View class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- DoVerb
- InsertAfter
- MakeNamedStyle
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: Panel class

An abstract class that comprises the BodyPanel, RepeatingPanel, HeaderFooterPanel, and SummaryPanel classes.

Contained by

Each instance of a Panel object is identified from its parent through an expanded property. The following classes can contain a Panel object:

Class
ChartView
Crosstab
Envelope
Form
FormLetter
MailingLabels
Report
Worksheet

Usage

The Panel class is an abstract class, so you cannot create an instance of a Panel. You can, however, represent all the classes of the Panel class, such as the BodyPanel class, with the Panel class. For example, you can write a sub that takes Panel as a parameter, so you can pass any instance of a Panel class to that sub.

Panel class members

Properties

Background AS Background class
Border AS Border class
Height
NamedStyle
Parent AS View class
Type

Methods

MakeNamedStyle

Events

None

Approach: PicturePlus class

A display element containing an image or an OLE object.

Note OLE objects are not supported under OS/2.

Contained by

Each instance of a PicturePlus object is identified from its parent through an expanded property. The following classes can contain a PicturePlus object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

A PicturePlus display element can serve the following functions:

- Display or edit images.
- Display or edit OLE objects.

Configure some important characteristics of a PicturePlus display element using the following properties:

- Allow freehand drawing with the mouse on top of a graphic or OLE object using the AllowDrawing property.
- Crop the image to the dimension of the field, or shrink the whole picture to fit in the field using the Display property.
- Stretch an image to fit the dimension of the field box using the Stretch property.

PicturePlus class members

Properties

- AllowDrawing
- Background AS Background class
- Border AS Border class
- DataField
- DataTable
- Display
- Height
- Left
- MacroDataChange
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- Position
- ReadOnly
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- Stretch
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: Picture class

An image displayed on a form, report, mailing label, form letter, envelope, or chart. The picture is not part of any record in the database; it is part of the view. The same picture appears for every record or page of records in the view.

Note Use the PicturePlus class to store images as part of records.

Contained by

Each instance of a Picture object is identified from its parent through an expanded property. The following classes can contain a Picture object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Pictures can enhance views of data.

Configure some important characteristics of a Picture display element using the following properties and methods:

- Specify the image you want to display using the SetPicture method.
- Set the background and shadow color of the Picture display element using the Background and ShadowColor properties.
- Set the position of the Picture display elements in the view by using the Left and Top properties.

Picture class members

Properties

- Background AS Background class
- Height
- Left
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus
- SetPicture

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: Query class

An SQL statement or table name which defines the data to retrieve through a Connection object.

Contained by

Class	Property
ResultSet	Query

Usage

The Query class is one of three classes that together allow you to access data in a batch process, bypassing the Approach user interface.

There are two strategies for defining a query:

- Retrieve an entire table using the TableName property.
- Retrieve a specific set of fields or records from a table by defining find conditions using the SQL property.

Using a Query object involves the following operations:

- Creating and opening a connection
- Creating a Query object using the New method
- Identifying the data to retrieve using the TableName or SQL properties
- Initializing and running a ResultSet object

If your query includes an SQL statement that does not produce a result set, call the Execute method from the Query object without initializing a ResultSet object.

Query class members

Properties

Connection AS Connection class
SQL
TableName

Methods

Execute
GetError
GetErrorMessage
GetExtendedErrorMessage
New

Events

None

Approach: RadioButton class

A display element indicating a mutually exclusive selection. A radio button can be on or off.

Contained by

Each instance of a RadioButton object is identified from its parent through an expanded property. The following classes can contain a RadioButton object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Radio buttons allow users to enter a value when you want them to enter only one of a limited set of values.

A RadioButton can be selected or not. If it is selected, the center is black.

Typically, radio buttons come in groups. When one radio button in the group is selected, the others become deselected.

Note You cannot create radio button groups using LotusScript, but you can set or unset the attribute of a radio button that is part of a group from Script.

Configure some important characteristics of a radio button using the following properties and methods:

- Set the values written to a field for each radio button state using the ClickedValue property.
- Set a default value for a new radio button using the SetState method.
- Set the label for the radio button using the LabelText property.

RadioButton class members

Properties

- Background AS Background class
- ClickedValue
- DataField
- DataTable
- Height
- IsClicked
- LabelAlignment
- LabelFont AS Font class
- LabelPosition
- LabelText
- Left
- MacroDataChange
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ReadOnly
- Relief
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Value
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus
- SetState

Events

- Change
- Click
- DoubleClick
- GotFocus
- LostFocus
- MouseDown
- MouseMove
- MouseUp

Approach: Rectangle class

A display element that you can add to a form, report, mailing label, form letter, envelope, or chart.

Contained by

Each instance of a Rectangle object is identified from its parent through an expanded property. The following classes can contain a Rectangle object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Rectangles can enhance views of data. For example, use a rectangle to group fields in a form.

Configure some important characteristics of a rectangle display element using the following properties and methods:

- Set the background and shadow color of the rectangle display elements using the Background and ShadowColor properties.
- Set the position of the rectangle display element in a group of overlapping objects by using the BringToFront and SendToBack methods.

Rectangle class members

Properties

- Background AS Background class
- Border AS Border class
- Height
- Left
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: RepeatingPanel class

In an Approach file that has joined databases, a repeating panel shows data from multiple records of a detail database, the records are all related to the current record of the main database.

Contained by

Each instance of a RepeatingPanel object is identified from its parent through an expanded property. The following classes can contain a RepeatingPanel object:

Class
Form

Usage

A repeating panel shows a one-to-many relationship between the records in two databases. Both databases must be joined in the Approach file (document). For example, create a repeating panel to show the relationship between a department and the employees in the department.

Configure some important characteristics of a repeating panel using the following properties and methods:

- Set the color of the repeating panel using the Background, Border and AlternateColors properties
- Determine the position of the panel on the form, including hidden forms, using the Left and Top properties.

RepeatingPanel class members

Properties

AlternateColors
Background AS Background class
Border AS Border class
Height
Left
MainTable
Name
NamedStyle
NumLines
Page
Parent AS View class
Top
Type
Width

Methods

MakeNamedStyle
New

Events

None

Approach: Report class

A Report object is an Approach report view.

Contained by

Each instance of a Report object is identified from its parent through an expanded property. The following classes can contain a Report object:

Class
Document

Usage

Configure some important characteristics of a report view using the following properties:

- Determine the document (.APR file) that contains the report using the Document property.
- Determine the number of columns that appear in the report using the NumColumns property.
- Keeps a record's fields on the same page when there is more than one line of fields in the record using the KeepRecsTogether property.

Report class members

Properties

Document AS Document class
KeepRecsTogether
MainTable
MenuBar
Name
NumColumns
ObjectList AS Collection class
OnSwitchFromMacro
OnSwitchToMacro
Parent AS Document class
TimerInterval
Type
Visible

Methods

New

Events

SwitchFrom
SwitchTo
UserTimer

Approach: ResultSet class

A set of records accessible outside the Approach user interface.

Data in a ResultSet can be edited; the edits are reflected in the original table.

Contained by

Class
None

Usage

The ResultSet class is one of three classes that together allow you to access data in a batch process, bypassing the Approach user interface. A result set acts just like a found set, but is available only through script.

You can create a result set in one of two ways:

- Convert a Table object already associated with an .APR file into a ResultSet object using the CreateResultSet method from the Table class.
- Open a connection to an existing table and define a query to retrieve data from that table to populate the result set.

Creating a result set though a connection involves the following operations:

- Creating and opening a connection
- Creating a query
- Initializing the ResultSet object using the New method
- Associating the ResultSet object using the Query property
- Creating the result set using the Execute method

A ResultSet object is a table that can be used whereever a Table object is used.

You can perform operations with the data such as the following:

- Determine the field size, name, and type.
- Navigate between fields, rows, or columns in a record or group of records.
- Edit values in fields.

Determine the commit behavior of the result set through the associated Connection object.

After making changes to data in a result set, use the UpdateRow method to commit the changes to the table.

ResultSet class members

Properties

- CurrentRow
- IsBeginOfData
- IsEndOfData
- IsReadOnly
- IsResultSetAvailable
- Query AS Query class

Methods

- AddRow
- Close
- DeleteRow
- Execute
- FieldExpectedDataType
- FieldID
- FieldName
- FieldNativeDataType
- FieldSize
- FirstRow
- GetError
- GetErrorMessage
- GetExtendedErrorMessage
- GetParameter
- GetParameterName
- GetValue
- LastRow
- New
- NextRow
- NumColumns
- NumParameters
- NumRows
- Options
- PrevRow
- SetParameter
- SetValue
- UpdateRow

Events

None

Approach: RoundRect class

A display element that you can add to a form, report, mailing label, form letter, envelope, or chart. A rounded rectangle has rounded corners instead of right-angled corners.

Contained by

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Rounded rectangles can enhance views of data. For example, use a rounded rectangle to separate sections fields in a form.

Configure some important characteristics of a rounded rectangle display element using the following properties and methods:

- Set the background and shadow color of the rectangle object using the Background and ShadowColor properties.
- Set the position of the rectangle display element in a group of overlapping objects by using the BringToFront and SendToBack methods.

RoundRect class members

Properties

- Background AS Background class
- Border AS Border class
- Height
- Left
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ShadowColor AS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Top
- Type
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: Sort class

The fields used to sort records.

Contained by

Class	Property
DocWindow	FindSort

Usage

A Sort object consists of one or more fields that indicate on which fields to sort the found set of records or an entire table. The first field you add to the Sort object is the primary field by which to sort the records.

Defining and running a sort in LotusScript involves the following operations:

- Creating a Sort object using the New method
- Adding fields to the sort using the Add method
- Executing the sort using the DocWindow object FindSort method

Sort class members

Properties

None

Methods

Add
GetAt
GetCount
New

Events

None

Approach: SummaryPanel class

A summary panel displays fields or calculations for groups of records.

Contained by

Each instance of a SummaryPanel object is identified from its parent through an expanded property. The following classes can contain a SummaryPanel object:

Class
ChartView
Report

Usage

A summary panel shows calculations involving more than one record, or distinguishes between record groupings.

Configure some important characteristics of a summary panel using the following properties:

- Set the color of the summary panel using the Background, and Border properties
- Determine the printing behavior, such as the expansion or reduction of the size of the panel to accommodate the data, using the Expand and Reduce properties
- The records used in the group represented by the summary panel using the GroupByDataField and GroupByDataTable properties

SummaryPanel class members

Properties

- Alignment
- Background AS Background class
- Border AS Border class
- Expand
- GroupByDataField
- GroupByDataTable
- GroupByEvery
- Height
- Location
- Name
- NamedStyle
- PageBreak
- Parent AS View class
- Reduce
- Type
- Width

Methods

- MakeNamedStyle
- New

Events

None

Approach: Table class

A table accessed through Approach.

Contained by

Each instance of a Table object is identified from its parent through an expanded property. The following classes can contain a Table object:

Class
Document

Usage

Use a Table object to retrieve the following kinds of information about the table:

- The name and path of the database
- The database field names, sizes, and types
- The number of records in the database

Update a document with new data by converting a ResultSet object to a Table object using the ReplaceWithResultSet method.

Access data from a Table object without going through display elements in the Approach user interface by converting the table to a result set using the CreateResultSet method.

The Table object has uses in tables other than data tables. For example, access information about the calculated fields in a document through the CalcTable property of the Document object. This property identifies a Table object.

The VarTable property of the Document object also identifies a Table object.

Table class members

Properties

FieldNames
FileName
FullName
NumFields
NumRecords
Parent!SaveMark(`MemList');AL(`H_LAS_PARENT_PROPERTY_MEMDEF',1); AS Document
classH_LAS_DOCUMENT_CLASS>CLASSDEF
Path
TableName

Methods

CreateResultSet
GetFieldFormula
GetFieldOptions
GetFieldSize
GetFieldType
GetFuriganaField
GetFuriganaMode
GetImeMode
GetImeState
ReplaceWithResultSet
SetFurigana
SetIme

Events

None

Approach: TextBox class

A TextBox object displays text in a text block in an Approach view.

Contained by

Each instance of a TextBox object is identified from its parent through an expanded property. The following classes can contain a TextBox object:

Class
BodyPanel
HeaderFooterPanel
RepeatingPanel
SummaryPanel

Usage

Text blocks can enhance views or provide information to users. For example, use a text block to provide instructions for making selections from check boxes or radio buttons on a form.

Configure some important characteristics of a text box using the following properties:

- Determine how the text is aligned within the text block using the Alignment property.
- Determine the font used to display the text using the Font property.
- Set the amount of space between the lines within the text block using the LineSpacing property.

TextBox class members

Properties

- Alignment
- Background AS Background class
- Border AS Border class
- Font AS Font class
- Height
- Left
- LineSpacing
- MacroClick
- MacroTabIn
- MacroTabOut
- Name
- NamedStyle
- NonPrinting
- Page
- Parent AS View class
- ShadowColorAS Color class
- ShowInPreview
- SlideLeft
- SlideUp
- TabOrder
- TabStop
- Text
- Top
- Type
- Vertical
- Visible
- Width

Methods

- BringToFront
- InsertAfter
- MakeNamedStyle
- New
- Refresh
- SendToBack
- SetFocus

Events

- Click
- DoubleClick
- MouseDown
- MouseMove
- MouseUp

Approach: View class

The base class of the derived classes you work with to construct a user interface, for example, forms, reports, worksheets, and charts.

The following Approach classes inherit the members of the View class:

ChartView	Crosstab	Envelope
Form	FormLetter	MailingLabel
Report	Worksheet	

Contained by

Each instance of a View object is identified from its parent through an expanded property. The following classes can contain a View object:

Class
Document

Usage

The View class is an abstract class, so you cannot create an instance of View.

You can, however, represent all of its derived classes with the View class. For example, you can write a procedure that takes "Source As View" as a parameter. This allows you to use any instance of a derived View class in that procedure.

For example, you could have a procedure designed to adjust margins. You don't know what type of view will be passed to the procedures, so you use View as a parameter. This way you can use any instance of a derived View class in the procedure and the procedure will run.

View class members

Properties

Document AS Document class
MainTable
MenuBar
Name
OnSwitchFromMacro
OnSwitchToMacro
Parent AS View class
TimerInterval
Type
Visible

Methods

None

Events

SwitchFrom
SwitchTo
UserTimer

Approach: Window class

The window class is a base class for the creation of objects such as the ApplicationWindow and DocWindow.

Contained by

Each instance of a Window object is identified from its parent through an expanded property. The following classes can contain a Window object:

Class
Application
Document

Usage

The Window class is an abstract class. That is, you cannot create an instance of Window. You can, however, represent all of Window's subclasses with the Window class. For example, you can write a subroutine which takes Window as a parameter. This allows you to pass any instance of a Window subclass to that subroutine.

Window class members

Properties

None

Methods

Close
GetHandle
Maximize
Minimize
Restore

Events

None

Approach: Worksheet class

A Worksheet object is an Approach worksheet.

Contained by

Each instance of a Worksheet object is identified from its parent through an expanded property. The following classes can contain a Worksheet object:

Class
Document

Usage

Configure some important characteristics of a worksheet using the following properties and methods:

- Determine the document that contains the worksheet using the Document property.
- Determine the main table used in the worksheet using the MainTable property.
- Add and delete columns in the worksheet using the AddColumn and RemoveColumn methods.

Worksheet class members

Properties

- Document AS Document class
- MainTable
- MenuBar
- Name
- OnSwitchFromMacro
- OnSwitchToMacro
- Parent AS View class
- PrintDate
- PrintPageNum
- PrintTitle
- TimerInterval
- Title
- Type
- Visible

Methods

- AddColumn
- GetText
- New
- RemoveColumn
- SetCellFocus
- SetText

Events

- CellDataChange
- CellGetFocus
- CellLostFocus
- SelectColumn
- SwitchFrom
- SwitchTo
- UserTimer

Approach: AutoCommit property

{button ,AL(`H_LAS_CONNECTION_CLASS',0)} See list of classes

Sets or returns the commit status of a SQL database connection.

Data type

Integer

Syntax

connectionobject.**AutoCommit** = *integer*

integer = *connectionobject*.**AutoCommit**

Parameters

None

Legal values

Value	Description
TRUE	(Default) Approach commits record changes, additions, or deletions directly to a table.
FALSE	Approach does not automatically commit changes, additions, or deletions. Changes must be committed manually, or AutoCommit must be set to TRUE.

Usage

By default, Approach displays data directly from the tables associated with an Approach application. When you change data through the user interface or in a ResultSet object, the changes are committed to the table after any of the following operations:

- Pressing ENTER
- Changing views or environments
- Running a macro
- Calling UpdateRow method

To change Approach's behavior so that changes are not made automatically to the table, set the AutoCommit property to FALSE. When you are ready to commit changes to the source database or rollback the changes, call the Transactions method.

When you change from manual-commit mode to auto-commit mode, you commit any open edits.

Set AutoCommit after executing the connection with the ConnectTo method.

Approach: CalcTable property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

(Read-only) Returns information for all calculated fields defined for the document (.APR file).

Data type

Table

Syntax

Set *calctable* = *documentobject*.**CalcTable**

Legal values

A Table object. If there are no calculated fields for a document, the table is empty.

Usage

You can add or remove calculated fields in the table using the CreateCalcField and DeleteCalcField methods.

Approach: CreateDate property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_CREATEDATE_EXSCRIPT',1)} See example

(Read-only) Returns the date and time the document (.APR file) was created.

Data type

Variant

Syntax

date/time = *documentobject*.**CreateDate**

Legal values

Any date or time.

The default values are the date and time the document was created.

Approach: CurrentFind property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

(Read-only) Returns the Find object most recently used in the document window.

Data type

Find object

Syntax

Set *findobject* = *docwindowobject*.**CurrentFind**

Parameters

None

Legal values

Any existing Find object.

Usage

Retrieve the object that defines the find conditions most recently used in a document (.APR file). To extract the conditions from the Find object, use the GetAt and GetCount methods.

Approach: CurrentRecord property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_CURRENTRECORD_EXSCRIPT',1)} See example

Sets or returns the record number of the current record.

Data type

Long

Syntax

docwindowobject.**CurrentRecord** = *recordnumber*

recordnumber = *docwindowobject*.**CurrentRecord**

Legal values

Any integer between 1 and the total number of records in the database.

Approach: CurrentSelection property

{button ,AL(`H_LAS_LISTBOX_CLASS;',0)} See list of classes

Sets or returns the item selected in a list box.

Data type

Integer

Syntax

integer = *listboxobject*.**CurrentSelection**

*listboxobject.***CurrentSelection** = *integer*

Parameters

None

Legal values

An integer indicating an item in the list box. The first item in the list is numbered zero.

Usage

This property presets the value shown in a list box.

When the list box object is not bound to a field, use this property to determine the value selected in the list box by the user.

Determine the total number of items in the list using the Count property.

Approach: CurrentSort property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

(Read-only) Returns the Sort object most recently used in the document window.

Data type

Sort object

Syntax

Set *sortobject* = *docwindowobject*.**CurrentSort**

Parameters

None

Legal values

Any existing Sort object.

Usage

Retrieve the object that defines the sort conditions for a document (.APR file). To extract the conditions from the Sort object, use the GetAt and GetCount methods.

Approach: Description property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_DESCRIPTION_EXSCRIPT',1)} See example

Sets or returns the description of the document (.APR file).

Data type

String

Syntax

documentobject.**Description** = *string*

string = *documentobject*.**Description**

Legal values

Any string up to 256 characters.

The default value for the Description property is Blank.

Usage

Use this property to read, enter, or modify the description listed for a document (.APR file). In the user interface, you can view the description by choosing File - Approach File Properties.

Enter a description for the document when you create a new document, or update an existing document.

Approach: Dispatch property

{button ,AL(`H_LAS_OLEOBJECT_CLASS;',0)} See list of classes

Note Dispatch is not supported under OS/2.

(Read-only) Returns an OLE object, if the OLE object supports OLE Automation.

You can directly access the properties and methods for the OLE control through Approach.

Data type

Variant

Syntax

Set *objectvariable* = *oleobject*.**Dispatch**

Legal values

Any OLE object property or method.

Usage

For example, you might have a word processor document embedded in an Approach form. If the word processor supports OLE Automation, you can pass text from Approach to the word processor's dictionary and thesaurus, and receive appropriate spellings or synonyms.

Approach: Display property

{button ,AL(`H_LAS_PICTUREPLUS_CLASS;',0)} See list of classes

Sets or returns the display behavior of the image in PicturePlus fields.

Data type

Long

Syntax

pictureplusobject.**Display** = *value*

value = *pictureplusobject*.**Display**

Legal values

Value	Description
$LtsPictureDisplayCrop	(Default) Crop the picture to the dimension of the field.
$LtsPictureDisplayShrink	Shrink the whole picture to fit in the field.

Usage

If the image is too large to fit within the field but you only want to display part of the image, crop it. If the image is too large to fit within the field and you still want to display the whole image, shrink it to fit.

Approach: ExcludeFirst property

{button ,AL(`H_LAS_FINDDUPLICATE_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_EXCLUDEFIRST_EXSCRIPT',1)} See example

Sets or returns whether all records found are returned in the found set when executing a FindDuplicate object.

Data type

Integer

Syntax

findduplicateobject.**ExcludeFirst** = *integer*

integer = *findduplicateobject*.**ExcludeFirst**

Parameters

None

Legal values

Value	Description
TRUE	The first record found by Approach in a set of duplicated records is not included in the found set.
FALSE	All records found by Approach are included in the found set.

Usage

Set this property to TRUE when you want to modify or delete duplicate records, but you want to leave the first instance of the duplicate record untouched.

Approach: FieldNames property

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_FIELDNAMES_EXSCRIPT',1)} See example

(Read-only) Returns a list of the fields defined in the specified table.

Data type

Variant

Syntax

string = *tableobject*.**FieldNames**(*n*)

Usage

This property identifies a list of field names. Use the property to specify a field as an argument in the following methods:

- GetFieldFormula
- GetFieldOptions
- GetFieldSize
- GetFieldType

To retrieve field names, step through the list by specifying an index to the list. For example, the following script prints the names of all of the fields in a table:

```
Sub FieldList(TableObject As Table)
'Print a list of the fields of the Table object that is passed in.
   Dim I As Integer
   'Loop from 0 to the number of fields in the table.
   For I = 0 To TableObject.NumFields - 1
      Print "Field name: " & TableObject.FieldNames(I)
   Next
End Sub
```

Approach: FileName property

{button ,AL(`H_LAS_DOCUMENT_CLASS;H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_FILENAME_EXSCRIPT',1)} See example

(Read-only) Returns the file name of the specified document (.APR file) or table.

Data type

String

Syntax

string = *documentobject*.**FileName**

or

string = *tableobject*.**FileName**

Legal values

The file name (and extension if it applies) of the specified document or table.

Usage

This property returns the file name and extension only of document and table files. No extension is returned if the table does not have one, such as a DB2 or Notes table.

To retrieve the full path name, use the FullName or Path properties.

Approach: FindSpecial property

{button ,AL(`H_LAS_FIND_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_FINDSPECIAL_EXSCRIPT',1)} See example

Sets or returns an existing FindDistinct, FindDuplicate, or FindTopLowest object used to define the Find object.

Data type

FindDistinct, FindDuplicate, or FindTopLowest object

Syntax

Set *findobject*.**FindSpecial** = *findspecialobject*

Set *findspecialobject* = *findobject*.**FindSpecial**

Parameters

None

Legal values

Any existing FindDistinct, FindDuplicate, or FindTopLowest object.

Usage

Use FindSpecial to append an existing FindDistinct, FindDuplicate, or FindTopLowest object as another condition of the Find object. The find conditions defined by the FindDistinct, FindDuplicate, or FindTopLowest object are applied to the results of the other find conditions specified in the Find object.

Approach: FullName property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_DOCUMENT_CLASS;H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_FULLNAME_EXSCRIPT',1)} See example

(Read-only) Returns the path of the current Approach executable (.EXE file), document (.APR file), or table.

Data type

String

Syntax

string = *object*.**Fullname**

Legal values

Object	Value	Description
Application object	*drive:**directory**filename.exe*	Drive and directory where the Approach executable file resides.
Document object	*drive:**directory**filename.apr*	Drive and directory where the Approach document file resides.
Table object	*drive:**directory**table.extension*	Drive and directory where the table file resides (non-SQL tables).
Table object	*owner**table*	Owner ID and table name of a table accessed through an ODBC server.

Usage

This property determines the full path. If you need only the file name of the document or table, use the FileName property.

Approach: GroupByDataField property

{button ,AL(`H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

{button ,AL(`H_las_GROUPBYDATAFIELD_EXSCRIPT',1)} See example

Sets or returns the field that records are grouped by to summarize data in a report.

Data type

String

Syntax

summarypanel.**GroupByDataField** = *stringexp*

stringexp = *summarypanel*.**GroupByDataField**

Legal values

Any field in any table (database file) of the document (.APR file), including calculated fields.

Usage

Determine which field is used to summarize data so you can generate an appropriate report. For example, you can create a report showing how many rooms were reserved on different dates by grouping on the Date Reserved field.

When you create a new report, you should also set the data table that contains the field you want to use.

Approach: GroupByDataTable property

{button ,AL(`H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

{button ,AL(`H_las_GROUPBYDATATABLE_EXSCRIPT',1)} See example

Sets or returns the table that contains the field that records are grouped by to summarize data in a report.

Data type

String

Syntax

summarypanel.**GroupByDataTable** = *stringexp*

stringexp = *summarypanel*.**GroupByDataTable**

Legal values

Any table (database file) of the document (.APR file).

Usage

Determine the table that contains the field you want to use to summarize data so you can generate an appropriate report. For example, you can create a report showing how many rooms were reserved on different dates by grouping on the Date Reserved field in the Schedule table.

Approach: GroupByEvery property

{button ,AL(`H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

Sets or returns the number of records you want to group in summary calculations.

Data type

Integer

Syntax

summarypanelobject.**GroupByEvery** = *value*

value = *summarypanelobject*.**GroupByEvery**

Legal values

Any number between 1 and the total number of records in the table (database file). To get the total number of records in the table, use the NumRecords property.

Usage

If the field you group by displays many records in the report, you can summarize a fixed number of records so they fit on a page.

Approach: IconBarVisible property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_ICONBARVISIBLE_EXSCRIPT',1)} See example

Sets or returns whether a set of SmartIcons is visible.

Data type

Integer

Syntax

docwindowobject.**IconBarVisible** = *flag*

flag = *docwindowobject*.**IconBarVisible**

Legal values

Value	Description
TRUE	(Default) A set of SmartIcons is visible.
FALSE	Not set of SmartIcons is visible.

Approach: IsClicked property

{button ,AL(`H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

(Read-only) Returns the state of a radio button.

Note The SetState method turns the radio button on or off.

Data type

Integer

Syntax

flag = *radiobuttonobject*.**IsClicked**

Legal values

Value	Description
TRUE	The radio button is selected.
FALSE	The radio button is not selected.

Usage

This property determines the state of a radio button, especially when the radio button is an unbound control and does not represent field values.

Determine if a radio button is clicked, and then perform another task based on the result. For example, if users click the "Nonsmoking" radio button, display only the list of available rooms that are designated "Nonsmoking."

Approach: KeepRecsTogether property

{button ,AL(`H_LAS_REPORT_CLASS;',0)} See list of classes

Sets or returns the whether to keep a record's fields on the same page when there is more than one line of fields in the record.

Data type

Integer

Syntax

flag = *reportobject*.**KeepRecsTogether**

reportobject.**KeepRecsTogether** = *flag*

Approach: Keywords property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_KEYWORDS_EXSCRIPT',1)} See example

Sets or returns the list of keywords that describe the document (.APR file).

Data type

String

Syntax

documentobject.**Keywords** = *string*

string = *documentobject*.**Keywords**

Legal values

Any string up to 256 characters.

Usage

Use this property to read, enter, or modify the keywords listed for a document (.APR file). In the user interface, you can view these keywords by choosing File - Approach File Properties.

Approach: Language property

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

(Read-only) Returns the language used in the current application as a 2-character code.

For example, English is EN, French is FR.

Data type

String

Syntax

string = *applicationobject*.**Language**

Legal values

All the 2-character language codes.

Usage

Determine which language is in use, so you can supply instructions or display views in the appropriate language.

Approach: LastModified property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_LASTMODIFIED_EXSCRIPT',1)} See example

(Read-only) Returns the date and time the document (.APR file) was last modified.

Data type

Variant

Syntax

variant = *documentobject*.**LastModified**

Legal values

The date and time the document was last modified.

Approach: Location property

{button ,AL(`H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

Sets or returns whether the summary panel in a report is a leading or trailing summary panel.

Data type

Long

Syntax

panelname.**Location** = *value*

value = *panelname*.**Location**

Legal values

Value	Description
aprSummaryLeading	The summary is a leading summary panel.
aprSummaryTrailing	The summary is a trailing summary panel

Usage

Display a leading summary panel when you want the summary to appear above the individual records you summarized. For example, in the Meeting Room Scheduler Smartmaster report, the room number appears before its associated reservation information.

Display a trailing summary panel when you want the summary to appear below the individual records you summarized. For example, in the Order Management SmartMaster Inventory report, the total number of items sold appears below its associated sale information.

Approach: LPObject property

{button ,AL(`H_LAS_OLEOBJECT_CLASS;',0)} See list of classes

Note LPObject is not supported under OS/2.

(Read only) Returns the value of the C pointer to the OleObject interface of the OLE object, or 0 if there is no connection to the OLE object yet.

Data type

Long

Syntax

value = *oleobject*.**LPObject**

Usage

Determine the address of the OLE object, and pass it to Dynamic Link Libraries.

The lock on the OLE object is not incremented when this property is accessed (that is, AddRef has not been called).

Approach: MainTable property

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPEATINGPANEL_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_MAINTABLE_EXSCRIPT',1)} See example

Sets or returns the main table (database file) for the view or repeating panel.

Data type

String

Syntax

object.**MainTable** = *string*

string = *object*.**MainTable**

Legal values

Any table in the document (.APR file).

Usage

If you have joined tables in an Approach file, one of the tables must be the main table for the view. A view displays each record from its main table. For example, an invoice view would use an invoice table as its main table.

A repeating panel must have a main table that provides the framework of records. The main table of the repeating panel must be one of the detail databases of the form it is displayed on. Each line in a repeating panel displays a record from its main table. For example, a repeating panel in a department form might list all the employees in the department.

Approach: MenuBar property

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_MENUBAR_EXSCRIPT',1)} See example

Sets or returns the type of menu displayed in the current view.

Data type

String

Syntax

viewobject.**MenuBar** = *string*

string = *viewobject*.**MenuBar**

Legal values

Any of the menus available for the document (.APR file), such as Default menu, Short menu, or the name of a custom menu.

Approach: Menus property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_MENUS_EXSCRIPT',1)} See example

(Read-only) Returns an array of menu names for the document (.APR file).

Data type

Variant

Syntax

array = *documentobject*.**Menus**

Legal values

All the names of the user-defined custom menus.

Approach: Modified property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_MODIFIED_EXSCRIPT',1)} See example

Sets or returns whether the document (.APR file) has been modified.

Data type

Integer

Syntax

flag = *documentobject*.**Modified**

Legal values

Value	Description
TRUE	The document has been modified.
FALSE	The document has not been modified.

Usage

Retrieve the value of the Modified property to determine if any changes have been made to a document after the last time it was saved.

Setting the Modified property to FALSE does the following, until another change is made to the document:

- Disables the "Save Approach File" menu command.
- Disables the user prompt to save changes to the document.

If you quit the document before other changes are made, Approach discards all changes made to the document since the last save.

When you make a change to the document, Approach resets Modified to TRUE.

Approach: NamedFindSorts property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NAMEDFINDSORTS_EXSCRIPT',1)} See example

(Read-only) Returns a list of the Named Find/Sorts in the document (.APR file).

Data type

Variant

Syntax

stringarray = *documentobject*.**NamedFindSorts**

Legal values

All the Named Find/Sorts in the document.

Approach: NamedFindSort property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_NAMEDFINDSORT_EXSCRIPT',1)} See example

Sets or returns the name of the current named find/sort.

Data type

String

Syntax

docwindowobject.**NamedFindSort** = *stringexp*

stringexp = *docwindowobject*.**NamedFindSort**

Legal values

Any of the named find/sorts in the document (.APR file).

Approach: NamedStyles property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NAMEDSTYLES_EXSCRIPT',1)} See example

(Read-only) Returns a list of the named styles in the document (.APR file).

Data type

Variant

Syntax

stringarray = *documenrobject*.**NamedStyles**

Legal values

All the named styles in the document.

Approach: NumColumns property

{button ,AL(`H_LAS_REPORT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMCOLUMNS_EXSCRIPT',1)} See example

Sets or returns the number of columns displayed in the current report.

Data type

Integer

Syntax

reportobject.**NumColumns** = *value*

value = *reportobject*.**NumColumns**

Usage

If the fields on the report don't take up much room left to right but run onto many pages, increase the number of columns on the report to use fewer pages.

Approach: NumFields property

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_NUMFIELDS_EXSCRIPT',1)} See example

(Read-only) Returns the number of fields in the table.

Data type

Integer

Syntax

integer = *tableobject*.**NumFields**

Usage

Use this method as the upper bound for a loop cycling through all of the fields in a table. The array of fields is zero-based, so the loop starts at zero and ends at *tableobject*.NumFields - 1.

Approach: NumJoins property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMJOINS_EXSCRIPT',1)} See example

(Read-only) Returns the number of joins in the document (.APR file).

Data type

Integer

Syntax

value = *documentobject*.**NumJoins**

Legal Values

Any number from 1 through 256.

Approach: NumLines property

{button ,AL(`H_LAS_REPEATINGPANEL_CLASS;',0)} See list of classes

Sets or returns the number of lines in a repeating panel.

Data type

Integer

Syntax

repeatingpanel.**NumLines** = *value*

value = *repeatingpanel*.**NumLines**

Legal values

Any integer from 1 through 30.

Usage

Determine the number of lines in a new repeating panel, or add more lines to an existing repeating panel, so you can display more data from the detailed database.

Approach: NumPages property

{button ,AL(`H_LAS_FORM_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMPAGES_EXSCRIPT',1)} See example

(Read-only) Returns the number of pages of a multi-page form or form letter.

Data type

Integer

Syntax

value = *formobject*.**NumPages**

or

value = *formletterobject*.**NumPages**

Legal values

The number of pages in the form or form letter.

To determine which page users are viewing, use the CurrentPageNum property.

Approach: NumRecordsFound property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMRECORDSFOUND_EXSCRIPT',1)} See example

(Read-only) Returns the number of records in a found set.

Data type

Long

Syntax

value = *docwindowobject*.**NumRecordsFound**

Legal values

Any number between 1 and the number of records in the main table (database file) for the view.

To avoid taking a long time to return the number of records, this method returns -1 if the table is from a SQL data source type and the number of records is not already known to Approach. Use the CountRecords method to retrieve the actual number.

Approach: NumRecords property

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_LAS_NUMRECORDS_EXSCRIPT',1)} See example

(Read-only) Returns the number of records in the table.

Data type

Long

Syntax

long = *tableobject*.**NumRecords**

Legal values

Any whole number.

Usage

To avoid taking a long time to return the number of records, this method returns -1 if the table is from a SQL data source type and the number of records is not already known to Approach. Use the ResultSet class NumRows method or the DocWindow class CountRecords method to retrieve the actual number.

Approach: NumRevisions property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMREVISIONS_EXSCRIPT',1)} See example

(Read-only) Returns the number of times the document (.APR file) has been updated.

Data type

Long

Syntax

value = *documentobject*.**NumRevisions**

Legal values

The number of times the document has been revised.

Usage

Use this property to determine how many times the document has changed since you last modified it.

In the user interface, you can view number of revisions by choosing File - Approach File Properties.

Approach: NumTables property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMTABLES_EXSCRIPT',1)} See example

(Read-only) Returns the number of tables (database files) in the document (.APR file).

Data type

Integer

Syntax

value = *documentobject*.**NumTables**

Legal values

The number of tables in the document.

Approach: NumViews property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_NUMVIEWS_EXSCRIPT',1)} See example

(Read-only) Returns the number of views in the document (.APR file).

Data type

Integer

Syntax

value = *documentobject*.**NumViews**

Legal values

The number of views in the document.

Approach: ObjectList property

{button ,AL(`H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;',0)} See list of classes

(Read-only) Returns a collection of all the display elements in the view.

Data type

Collection

Syntax

Set *collection* = *view*.**ObjectList**

Legal values

The list of display elements in the document (.APR file).

Approach: OnSwitchFromMacro property

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_ONSWITCHFROMMACRO_EXSCRIPT',1)} See example

Sets or returns the name of the macro to run when users switch from the current view.

Data type

String

Syntax

viewobject.**OnSwitchFromMacro =** *string*

string = *viewobject*.**OnSwitchFromMacro**

Legal values

A string whose value is the name of a macro in the document (.APR file).

Approach: OnSwitchToMacro property

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_ONSWITCHTOMACRO_EXSCRIPT',1)} See example

Sets or returns the name of the macro to run when users switch to this view.

Data type

String

Syntax

viewobject.**OnSwitchToMacro =** *string*

string = *viewobject*.**OnSwitchToMacro**

Legal values

A string whose value is the name of a macro in the document (.APR file).

Approach: PageBreak property

{button ,AL(`H_LAS_SUMMARYPANEL_CLASS;',0)} See list of classes

{button ,AL(`H_las_PAGEBREAK_EXSCRIPT',1)} See example

Sets or returns a page break in a report after each summary group.

Data type

Integer

Syntax

summarypanel.**PageBreak =** *flag*

flag = *summarypanel*.**PageBreak**

Legal values

Value	Description
TRUE	Insert a page break.
FALSE	Do not insert a page break.

Usage

Insert a page break when you want to start a new summary group on a new page. For example, in a report showing product sales grouped by month, insert a page break so each month's data appears on a new page.

Approach: Password property

{button ,AL(`H_LAS_CONNECTION_CLASS;',0)} See list of classes

{button ,AL(`H_las_PASSWORD_EXSCRIPT',1)} See example

(Write-only) Sets a password used for connecting to a server.

If a password is required and you do not provide it, the Logon dialog appears.

Data type

String

Syntax

connectionobject.**Password** = *string*

Legal values

Any string.

Approach: Path property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_DOCUMENT_CLASS;H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_PATH_EXSCRIPT',1)} See example

(Read-only) Returns the drive and directory of the application executable file (.EXE file), document (.APR file), or table.

Data type

String

Syntax

string = *object*.**Path**

Legal values

Object	Value	Description
Application object	*drive:**directory*\	A drive and directory where the Approach executable file resides.
Document object	*drive:**directory*\	A drive and directory where the Approach document file resides.
Table object	*drive:**directory*\	A drive and directory where the database file resides.
Table object	*owner*\	Owner of a table accessed through an ODBC server.

Usage

Returns the path only. If you want the path and file name use the FullName property. If you want the file name only, use the FileName property.

Approach: PrintDate property

{button ,AL(`H_LAS_CROSSTAB_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Sets or returns whether to print the date on a printed copy of a worksheet or crosstab.

Data type

Integer

Syntax

viewobject.**PrintDate =** *flag*

flag = *viewobject*.**PrintDate**

Legal values

Value	Description
TRUE	Print the date.
FALSE	Do not print the date.

Approach: PrintPageNum property

{button ,AL(`H_LAS_CROSSTAB_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Sets or returns whether to print the page number on a printed copy of a worksheet or crosstab.

Data type

Integer

Syntax

viewobject.**PrintPageNum =** *flag*

flag = *viewobject*.**PrintPageNum**

Legal values

Value	Description
TRUE	Print the page number.
FALSE	Do not print the page number.

Approach: PrintTitle property

{button ,AL(`H_LAS_CROSSTAB_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Sets or returns whether to print the title on a printed copy of a worksheet or crosstab.

Data type

Integer

Syntax

viewobject.**PrintTItle =** *flag*

flag = *viewobject*.**PrintTItle**

Legal values

Value	Description
TRUE	Print the title.
FALSE	Do not print the title.

Approach: Redraw property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_REDRAW_EXSCRIPT',1)} See example

Sets or returns whether the screen is redrawn each time a change occurs in the user interface.

Data type

Integer

Syntax

docwindowobject.**Redraw =** *flag*

flag = *docwindowobject*.**Redraw**

Legal values

Value	Description
TRUE	(Default) Redraw the screen.
FALSE	Do not redraw the screen.

Usage

Suppose your script calls for the addition and removale of many objects in the user interface. To reduce onscreen flickering, first set the Redraw property to FALSE; then add and remove the objects; reset the Redraw property to TRUE; and finally, call the Repaint method.

Approach: Selection property

{button ,AL(`H_LAS_APPLICATION_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;',0)} See list of classes

(Read-only) Returns the currently selected display element object in the view.

For example, if the user clicks a field box, the Selection property returns the corresponding FieldBox display element.

Data type

Variant

Syntax

Set *displayelementobject* = *viewobject*.**Selection**

Approach: ShowRelated property

{button ,AL(`H_LAS_CROSSTAB_CLASS;',0)} See list of classes

Sets or returns whether related rows and columns appear in crosstabs.

Data type

Integer

Syntax

viewobject.**ShowRelated =** *flag*

flag = *viewobject*.**ShowRelated**

Legal values

Value	Description
TRUE	Show related rows and columns in crosstabs.
FALSE	Do not show related rows and columns in crosstabs.

Usage

By default, a crosstab shows only subheaders with related records to the header. For example, nonzero values, by default, do not appear under headers. Use this property to show all subheaders even if no record values correspond to the header and subheader combination.

Approach: StatusBarVisible property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_STATUSBARVISIBLE_EXSCRIPT',1)} See example

Sets or returns whether the status bar is visible.

Data type

Integer

Syntax

docwindowobject.**StatusBarVisible =** *flag*

flag = *docwindowobject*.**StatusBarVisible**

Legal values

Value	Description
TRUE	(Default) The status bar is visible.
FALSE	The status bar is hidden.

Usage

Hide the status bar when you want to expand the size of a view, enhance its appearance, or prevent users from switching manually to another view or environment.

Approach: TableName property (Query class)

{button ,AL(`H_LAS_QUERY_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABLENAME_query_class_EXSCRIPT',1)} See example

Sets or returns the file name of the table specified by a Query object.

Data type

String

Syntax

queryobject.**TableName =** *string*

string = *queryobject*.**TableName**

Legal values

Any table name accessible from the connection specified by the Query's Connection property. The table name can include the file extension and path.

Usage

This property is mutually exclusive of the SQL property. When you specify an SQL statement, Approach executes the SQL statement and clears the TableName property setting.

Approach: TableName property (Table class)

{button ,AL(`H_LAS_TABLE_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABLENAME_TABLE_CLASS_EXSCRIPT',1)} See example

Returns the name of the table specified by a Table object.

Data type

String

Syntax

tableobject.**TableName =** *string*

string **=** *tableobject*.**TableName**

Legal values

The name of a table associated with a document.

If a table alias is assoicated with the document, TableName reports the alias name as the table name with an extension. For example, if the table Employees is joined to itself, TableName returns the names of the two instances of the table as follows:

```
Employees:1
Employees:2
```

If there are tables joined in the document with the same name, TableName returns the names of the two tables as follows:

```
Order_1
Order_2
```

Usage

For a Table object, retrieve the TableName for use as an input for other properties such as the GetTableByName method of a Document object.

Use the FileName, FullName, and Path properties to retrieve other table information.

Approach: Tables property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_TABLES_EXSCRIPT',1)} See example

(Read-only) Returns the tables used in the document (.APR file).

Data type

BaseCollection

Syntax

basecollectionobject = *documentobject*.**Tables**

Legal values

All the tables in the document.

Approach: TimerInterval property

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

{button ,AL(`H_las_TIMERINTERVAL_EXSCRIPT',1)} See example

Sets or returns the time duration of the user timer, in milliseconds.

Data type

Long

Syntax

viewobject.**TimerInterval =** *flag*

flag = *viewobject*.**TimerInterval**

Legal values

Any integer greater than or equal to 200.

Usage

This property sets the length of a timer. When the timer expires, the UserTimer event starts, and the timer resets to the TimerInterval value and counts down again. This cycle continues until you set the TimerInterval to zero.

Approach: Title property

{button ,AL(`H_LAS_CROSSTAB_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Sets or returns the title of the worksheet or crosstab.

Data type

String

Syntax

*viewname***.Title =** *stringexp*

stringexp = *viewname***.Title**

Legal values

You can set this property to any string up to 256 characters.

Approach: Transparent property

{button ,AL(`H_LAS_COLOR_CLASS;',0)} See list of classes

(Read-only) Returns whether the color is transparent.

Data type

Integer

Syntax

value = *color*.**Transparent**

Legal values

0 means the color is opaque and 255 means the color is transparent.

Usage

Whatever is beneath a transparent object shows through. This is useful, for example, for placing text inside a circle instead of a rectangle.

Approach: User property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_USER_EXSCRIPT',1)} See example

(Read-only) Returns the current user or group name who is using the document (.APR file).

Data type

String

Syntax

stringexp = *documentobject*.**User**

Legal Values

Any user or group that has access to the document.

Usage

You can use the name of the user or group to personalize the document, or refer to them by name when you are requesting input.

Approach: VarTable property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_VARTABLE_EXSCRIPT',1)} See example

(Read-only) Returns information for all variable fields defined in the document (.APR file).

Data type

Table

Syntax

Set *vartableobject* = *documentobject*.**VarTable**

Legal Values

The legal value for the VarTable property is VarTable.

The default value for the VarTable property is the variable table for the current document.

Approach: Vertical property

{button ,AL(`H_LAS_TEXTBOX_CLASS;',0)} See list of classes

This property is not used in this version of Approach.

Approach: Views property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_VIEWS_EXSCRIPT',1)} See example

(Read-only) Returns the names of all the views in the document (.APR file).

Data type

BaseCollection

Syntax

basecollection = *documentobject*.**Views**

Legal Values

The names of all the views in the current document.

Usage

Use this property to list all the views for a document (.APR file). In the user interface, you can see the list by choosing File - Approach File Properties.

Approach: ViewTabVisible property

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

{button ,AL(`H_las_VIEWTABVISIBLE_EXSCRIPT',1)} See example

Sets or returns whether the view tabs are visible.

Data type

Integer

Syntax

docwindowobject.**ViewTabVisible =** *flag*

flag = *docwindowobject*.**ViewTabVisible**

Legal values

Value	Description
TRUE	(Default) Display the view tabs.
FALSE	Do not display the view tabs.

Usage

Hide view tabs to expand the size of a view, enhance its appearance, or prevent users from switching manually to another view.

Approach: Windows property

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

{button ,AL(`H_las_WINDOWS_EXSCRIPT',1)} See example

(Read-only) Returns a list of all windows controlled by the current application (.EXE file).

Data type

BaseCollection

Syntax

Set *collectionobject* = *applicationobject.***Windows**

Legal values

All windows controlled by the application.

Usage

Use this property to identify windows open in Approach without having to know them explicitly by name.

Identify a single window by specifying its index in the Windows BaseCollection.

For example, if you want to close the first window opened in the current Approach session, specify the first window in Windows (the BaseCollection is numbered starting from zero):

```
Call CurrentApplication.Windows(0).Close()
```

Approach: Window property

{button ,AL(`H_LAS_DOCUMENT_CLASS;',0)} See list of classes

{button ,AL(`H_las_WINDOW_EXSCRIPT',1)} See example

(Read-only) Returns the document window for this document (.APR file).

Data type

DocWindow

Syntax

documentwindow = *documentobject*.**Window**

Legal Values

Any of the document windows in the current document.

Usage

Determine which document window is in use so you can update the characteristics of the window, such as which tool bars are visible.

Approach: Broadcast event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when you invoke the Approach application from one of the following:

- the operating system command line prompt
- an embedded Approach object
- another SmartSuite product script

Internal syntax

Broadcast(**source**, *parameter*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

parameter

A long that is passed from the command line to the Broadcast event script.

Usage

This event is a mechanism for passing information to Approach from the command line.

For example, to specify a predetermined task, invoke Approach with the following command, where the task is defined in the Broadcast event script for Test.APR:

```
Approach.exe "Test.APR" /BROADCAST=1
```

Approach: CellDataChange event

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when you change the data in a worksheet cell.

Internal syntax

CellDataChange(**source**, *columnlabel*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

columnlabel

A string representing the header of the column. The header, or label, may differ from the column's field name.

Approach: CellGetFocus event

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when a worksheet cell is selected by being tabbed into, clicked, or selected using the keyboard.

Internal syntax

CellGetFocus(**source**, *columnlabel*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

columnlabel

A string representing the header of the column. The header, or label, may differ from the column's field name.

Approach: CellLostFocus event

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when a worksheet cell is selected and then deselected by a user's tabbing out of it, clicking another cell or selecting another cell using the keyboard.

Internal syntax

CellLostFocus(**source**, *columnlabel*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

columnlabel

A string representing the header of the column. The header, or label, may differ from the column's field name.

Approach: Change event

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

Occurs when the value stored in a field changes. The change can be triggered in several ways:

- If the display element is a check box or radio button, clicking the object triggers the change event.
- If the user tabs to the display element and changes its value, the change event is triggered when the user changes the focus to another display element (by tabbing or clicking in the view or by moving to another record) or when the record is saved.

Internal syntax

Change(**source**)

Parameters

source

A LotusScript keyword representing the object that is associated with the event. Always use the word **source** as the parameter and not the current object's name.

Usage

Determine when the value in a field changes, so you can initiate another action.

For example, when users enter data in the Number of Children field box and change the focus to another display element, you can check the Family Rate check box.

Approach: Click event

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

Occurs when a display element is selected using the mouse, the keyboard, or a shortcut key.

Internal syntax

Click(**source***, x, y, flags*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the object's name.

x

A long that passes the horizontal coordinate of the click location, in twips, to the sub.

y

A long that passes the vertical coordinate of the click location, in twips, to the sub.

flags

A long representing which mouse button was clicked, the right or left.

Usage

Use when you want to initiate an action. For example, you can write a script to change the font color of a paragraph of text when the user clicks a command button.

Approach: CloseWindow event

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Occurs when a window in a document (.APR file) is closed.

Internal syntax

CloseWindow(**source***, document*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

document

The Document object.

Approach: DocumentClose event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when you close the active document window (.APR file).

Internal syntax

DocumentClosed(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Usage

When you or your users close a document, you can initiate another action, such as running a procedure or opening another document.

Approach: DocumentCreated event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when a new document (.APR file) is created in the application.

Internal syntax

DocumentCreated(**source**, *document*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

document

The Document object.

Usage

When you or your users create a new document, you can determine the settings for the document, such as which set of icons is visible.

Approach: DocumentOpened event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when you or a user opens an existing document (.APR file).

Internal syntax

DocumentOpened(**source**, *document*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

document

The Document object.

Usage

When you or your users open a document, you can initiate another action, such as moving the insertion point to a particular field in a specific record.

Approach: DoubleClick event

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

Occurs when a display element is double-clicked.

Internal syntax

DoubleClick(**source**, *x*, *y*, *flags*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the object's name.

x

A long that passes the horizontal coordinate of the click location, in twips, to the sub.

y

A long that passes the vertical coordinate of the click location, in twips, to the sub.

flags

A long representing which mouse button was clicked, the right or left.

Usage

Use when you want to initiate an action. For example, you can write a script to resize a rectangle when the user double-clicks it.

Approach: GotFocus event

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

Occurs when a display element is selected by being tabbed into, clicked, or selected using the keyboard. A display element that is selected can appear bold, highlighted with a dashed border or dark border.

Internal syntax

GotFocus(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Usage

Determine when users select a display element, and then initiate another action.

For example, when users tab into a field box, you can enable a set of radio buttons.

Approach: KeyDown event

{button ,AL(`H_LAS_FIELDBOX_CLASS;',0)} See list of classes

Occurs when a key is pressed down. It determines the status of every key on the keyboard, including ALT, CTRL, SHIFT, function keys, as well as a combination of keys.

Internal syntax

KeyDown(**source**, *charcode, repeats, flags, overridedefault*)

Parameters

source

A LotusScript keyword representing the display element that receives the event. Always use the word **source** as the parameter and not the current display element's name.

charcode

An integer passing the character code of the key the user pressed, such as KEY_F1 or KEY_ALT, to the subroutine. Use the ASCII character codes to determine which key is pressed.

repeats

An integer representing the number of times the keystroke is pressed.

flags

An integer representing the bit values for extended keys. For a list of valid flag values, see the following table.

Value	Description
0 -7	Scan code. Value depends on the original equipment manufacturer.
8	Extended key. For example, the right-hand ALT and CTRL keys of an enhanced 101- or 102-key keyboard. 1 = extended key. 0 = not an extended key.
9 -12	Reserved. Do not use.
13	Context code. 1 = user holds down ALT while pressing the key. 0 = user presses the key without holding down ALT.
14	Previous key state. 1 = key is pressed down before a message is sent. 0 = key is released before message is sent.
15	Transition state. 1 = key is being released. 0 = key is being pressed.

overridedefault

An integer representing whether the keystroke behaves according to its default setting or to another setting.

Usage

Initiates an action when a user presses a keystroke or combination of keystrokes.

For example, you can write a script that runs when the user presses the CTRL +C keys to copy a display element.

Approach: KeyPress event

{button ,AL(`H_LAS_FIELDBOX_CLASS;',0)} See list of classes

Occurs when a key is pressed on the keyboard. It only determines the status of printable characters such as A-Z, 0-9, and so on, and a few other keys such as the TAB and ENTER keys.

Internal syntax

KeyPress(**source**, *charcode, repeats, flags, overridedefault*)

Parameters

source

A LotusScript keyword representing the display element that receives the event. Always use the word **source** as the parameter and not the current display element's name.

charcode

An integer passing the character code of the key the user pressed, such as KEY_F1 or KEY_ALT, to the subroutine. Use the ASCII character codes to determine which key is pressed.

repeats

An integer representing the number of times the keystroke is pressed without releasing the key.

flags

A short representing the bit values for extended keys. For a list of valid flag values, see the following table.

Value	Description
0-7	Scan code. Value depends on the original equipment manufacturer.
8	Extended key. For example, the right-hand ALT and CTRL keys of an enhanced 101- or 102-key keyboard. 1 = extended key. 0 = not an extended key.
9-12	Reserved. Do not use.
13	Context code. 1 = user holds down ALT while pressing the key. 0 = user presses the key without holding down ALT.
14	Previous key state. 1 = key is pressed down before a message is sent. 0 = key is released before message is sent.
15	Transition state. 1 = key is being released. 0 = key is being pressed.

overridedefault

An integer representing whether the keystroke behaves according to its default setting or to another setting.

Usage

Initiates an action when a user presses a printable character key.

For example, you can write a script that determines what users are entering into a field box, and then send the text directly to the printer.

Approach: KeyUp event

{button ,AL(`H_LAS_FIELDBOX_CLASS;',0)} See list of classes

Occurs when a key is released. It determines the status of every key on the keyboard, including ALT, CTRL, SHIFT, function keys, as well as a combination of keys.

Internal syntax

KeyUp (**source**, *charcode, repeats, flags, overridedefault*)

Parameters

source

A LotusScript keyword representing the display element that receives the event. Always use the word **source** as the parameter and not the current display element's name.

charcode

An integer passing the character code of the key the user pressed, such as KEY_F1 or KEY_ALT, to the subroutine. Use the ASCII character codes to determine which key is pressed.

repeats

An integer representing the number of times the keystroke is pressed without releasing the key.

flags

A short representing the bit values for extended keys. For a list of valid flag values, see the following table.

Value	Description
0-7	Scan code. Value depends on the original equipment manufacturer.
8	Extended key. For example, the right-hand ALT and CTRL keys of an enhanced 101- or 102-key keyboard. 1 = extended key. 0 = not an extended key.
9-12	Reserved. Do not use.
13	Context code. 1 = user holds down ALT while pressing the key. 0 = user presses the key without holding down ALT.
14	Previous key state. 1 = key is pressed down before a message is sent. 0 = key is released before message is sent.
15	Transition state. 1 = key is being released. 0 = key is being pressed.

overridedefault

An integer representing whether the keystroke behaves according to its default setting or to another setting.

Usage

Initiates an action when a user releases a keystroke or combination of keystrokes.

For example, you can write a script that runs when the user releases the CTRL +C keys when copying a display element.

You can also detect when a key is released so you can cause something to happen until the key is released.

For example, if users hold down an arrow key to move a display element, use the KeyUp event to stop moving the display element.

Approach: LostFocus event

{button ,AL(`H_LAS_CHECKBOX_CLASS;H_LAS_DROPDOWNBOX_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LISTBOX_CLASS;H_LAS_RADIOBUTTON_CLASS;',0)} See list of classes

Occurs when a selected display element loses focus when users tab out of it, click another display element, or select another display element using the keyboard.

Internal syntax

LostFocus(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Usage

Determine when users have moved focus from a display element, and then initiate another action.

For example, when users tab out of a field box, you can enable a set of radio buttons.

Approach: MailCheck event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when the mail is checked from within Approach.

Internal syntax

MailCheck(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Approach: MailSend event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when mail is sent from within Approach.

Internal syntax

MailSend(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Approach: MouseDown event

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

Occurs when a mouse button is held down while the mouse pointer is on a display element.

Internal syntax

MouseDown(**source**, *x*, *y, flags)*

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the object's name.

x

A long that passes the horizontal coordinate of the click location, in twips, to the sub.

y

A long that passes the vertical coordinate of the click location, in twips, to the sub.

flags

A long representing which mouse button was clicked, the right or left.

Usage

Combine this event with the MouseUp event to monitor when the user drags a display element to a different location.

Approach: MouseMove event

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

Occurs when the mouse is moved while the mouse button is held down on a display element.

Internal syntax

MouseMove(**source**, *x*, *y, flags)*

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the object's name.

x

A long that passes the horizontal coordinate of the click location, in twips, to the sub.

y

A long that passes the vertical coordinate of the click location, in twips, to the sub.

flags

A long representing which mouse button was clicked, the right or left.

Usage

Combine this event with the MouseUp and MouseDown events to monitor when the user drags a display element to a different location.

Approach: MouseUp event

{button ,AL(`H_LAS_BUTTON_CLASS;H_LAS_CHECKBOX_CLASS;H_LAS_ELLIPSE_CLASS;H_LAS_FIELDBOX_CLASS;H_LAS_LINEOBJECT_CLASS;H_LAS_OLEOBJECT_CLASS;H_LAS_PICTUREPLUS_CLASS;H_LAS_PICTURE_CLASS;H_LAS_RADIOBUTTON_CLASS;H_LAS_RECTANGLE_CLASS;H_LAS_ROUNDRECT_CLASS;H_LAS_TEXTBOX_CLASS;',0)} See list of classes

Occurs when a mouse button is released while the mouse pointer is on a display element.

Internal syntax

MouseUp(**source**, *x*, *y, flags*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the object's name.

x

A long that passes the horizontal coordinate of the click location, in twips, to the sub.

y

A long that passes the vertical coordinate of the click location, in twips, to the sub.

flags

A long representing which mouse button was clicked, the right or left.

Usage

Combine this event with the MouseDown event to monitor when the user drags a display element to a different location.

Approach: NewRecord event

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Occurs when a new record is created.

Internal syntax

NewRecord(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Approach: OpenWindow event

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Occurs when a window in a document (.APR file) opens.

Internal syntax

OpenWindow(**source**, *document*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

document

The Document object.

Approach: PageSwitch event

{button ,AL(`H_LAS_FORM_CLASS;',0)} See list of classes

Occurs when users switch from one page to another in a form .

Internal syntax

PageSwitch(**source**, *pagefrom*, *pageto*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

pagefrom

A long representing the page from which you switched.

pageto

A long representing the page to which you switched.

Approach: Quit event

{button ,AL(`H_LAS_APPLICATION_CLASS;',0)} See list of classes

Occurs when you exit Approach.

Internal syntax

Quit(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Usage

When you or your users close the application, you can intitiate another action, such as returning all user preferences to the default settings.

Approach: RecordChange event

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Occurs when users change from one record to another.

Internal syntax

RecordChange(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Approach: RecordCommit event

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Occurs when a record is committed when users move to another record or add a new record.

Internal syntax

RecordCommit(**source**, *table*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

table

A string representing the name of a table in the document (.APR file).

Approach: SelectColumn event

{button ,AL(`H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when users select a worksheet column by tabbing into it, clicking it, or selecting it with the keyboard.

Internal syntax

SelectColumn(**source**, *columnlabel*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

columnlabel

A string representing the header of the column. The header, or label, may differ from the column's field name.

Approach: SelectionChange event

{button ,AL(`H_LAS_LISTBOX_CLASS;',0)} See list of classes

Occurs when a user changes the item selected in a list box.

Internal syntax

SelectionChange(**source**)

Parameters

source

A LotusScript keyword representing the object that is associated with the event. Always use the word **source** as the parameter and not the current object's name.

Usage

Determine when the value of a list box changes so you can initiate another action.

Approach: SwitchFrom event

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when you switch from one view to another view.

Internal syntax

SwitchFrom(**source**, *view*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

view

The View object from which you switched.

Approach: SwitchTo event

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when you switch to one view from another view.

Internal syntax

SwitchTo(**source**, *view*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

view

The View object to which you want to switch.

Approach: UserTimer event

{button ,AL(`H_LAS_CHARTVIEW_CLASS;H_LAS_CROSSTAB_CLASS;H_LAS_ENVELOPE_CLASS;H_LAS_FORMLETTER_CLASS;H_LAS_FORM_CLASS;H_LAS_MAILINGLABELS_CLASS;H_LAS_REPORT_CLASS;H_LAS_VIEW_CLASS;H_LAS_WORKSHEET_CLASS;',0)} See list of classes

Occurs when the user timer expires.

Internal syntax

UserTimer(**source**)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

Usage

Set the length of the timer with the TimerInterval property. When the timer expires, the UserTimer event starts, and the timer resets to the TimerInterval value and counts down again. This cycle continues until you set the TimerInterval to zero.

Approach: ViewSwitch event

{button ,AL(`H_LAS_DOCWINDOW_CLASS;',0)} See list of classes

Occurs when users switch from one view to another view within a document (.APR file).

Internal syntax

ViewSwitch(**source**, *fromview*, *toview*)

Parameters

source

A LotusScript keyword representing the object that receives the event. Always use the word **source** as the parameter and not the current object's name.

fromview

The View object from which the user switched.

toview

The View object to which the user switched.

action bar
A set of buttons located initially at the top of the Approach work area, below the SmartIcons. The set of buttons you see most often lets you

- Switch to the Browse or Design environment
- Create a new record
- Create a find request in the current view or the Find Assistant
- Execute a named find

To hide or move the bar, use the right mouse button to click the bar in a space between buttons; then choose a location from the menu.

You can also drag the action bar to a new location, or let it float.

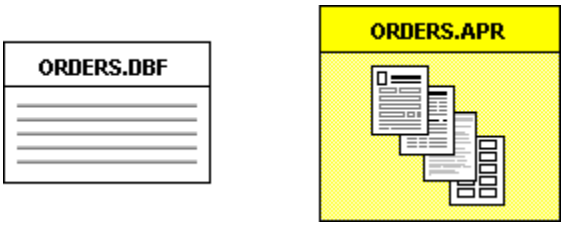
alias

A virtual copy of a database file, for use in special types of joins. An alias is not a physical duplicate of a database, but acts like it for the purposes of joining.

Use an alias to join a database to itself for advanced kinds of summaries and groupings, or to be able to join to a single database in multiple ways.

Approach file

The Approach file does not store data; instead, it stores the views you create. Through these views, you can look at and work with your data, which is stored in the database file(s) associated with the Approach file.



An Approach file can contain as many views (forms, reports, worksheets, and so on) as you need for your database application. The Approach file also stores calculated fields, variable fields, macros, and scripts.

When you create or open a database file, Approach automatically creates an Approach file for it.

Approach file extension: .APR

Approach file password

A sequence of characters you must enter before you can work in an Approach file.

- Passwords can have up to 16 characters.
- They are not case-sensitive.

One file can have more than one password. Different passwords for the same file can give you different privileges in that file.

For example, one password can give you complete access to edit and enter data in the associated databases as well as change the design of the views. Another password for the same file might let you do nothing except look at data. This is useful for files on networks; such files often must be available to many users, not all of whom should be able to change the file.

To define a password for an Approach file, choose File - TeamSecurity.

You can also define passwords for dBASE and FoxPro database files associated with the Approach file.

scrolling list
Three kinds of data-entry types for fields offer a predefined list of values, from which users select one value. When the predefined list is long enough, Approach adds a vertical scroll bar to the list. The following data-entry types are scrolling lists:

- Drop-down box
- Field box and list
- List box

arithmetic expression

An expression that performs a basic calculation on numeric, date, or time values. For example:

ShipDate + 15 returns a date equal to 15 days from the date in the ShipDate field of the current record.

Operators used in arithmetic expressions:

+	Addition
-	Subtraction
*	Multiplication
/	Division

ascending order
The sort order that arranges records from

- A to Z, for text (usually case-insensitive)
- Smallest to largest, for numbers
- Earliest to latest, for dates and times

Boolean field

A field that stores a single value, either Yes or No. You can enter these values in a Boolean field:

For Yes	For No
Yes, yes	No, no
Y, y	N, n
1	0

If you enter any other values (even, for example, False), Approach returns Yes.

A check box, which you define to have a checked and an unchecked value, is a good data-entry type to use with a Boolean field.

Browse

The environment in Approach for entering, editing, and viewing data in a database.

To go to Browse, do one of the following:

- Click the Browse button in the action bar.
- Click the Environment button in the status bar and select Browse.
- From the View menu, choose Browse & Data Entry.
- Press CTRL+B.

Any changes you make to records or any new records you create are saved automatically by Approach.

calculated field

A field that stores a formula using data from a record.

Enter the formula as part of the field definition. Write the formula to calculate with data

- In one record at a time.
- Across a range of records, using a summary function. For example, SCount(Account_Name).

After you add the calculated field to a view, Approach displays the result of the formula for each record. If the formula uses a summary function, go to Print Preview or Design to see the result.

Calculated fields are stored in the Approach file (.APR), not the database. They appear at the bottom of the Field Definition list and, in italics, at the bottom of the Add Field list.

check box

A data-entry type for fields. In Browse, to enter data in the field, you must select the check box or deselect it.

Usually, a field is represented by only one check box. A check box is especially useful as a data-entry type for a Boolean field, which accepts only Yes or No as values.

To make a field a check box: In Design, double-click the field to display the InfoBox. In the Basics tab, select Check boxes as the data-entry type. Define one checked value, one unchecked value, and a label for each check box.

When you first add a new field to a view and make it a check box, it appears to be deselected (no check mark). Actually, it has a null value--that is, the field has neither the checked value nor the unchecked value in it. In the Default Value tab of the Field Definition dialog box, you can define the checked or unchecked value as default data. This is called initializing the check box.

client

- In network terminology, a computer used to gain access to files or applications on a network.
- In OLE terminology, an application that receives data from a server application.

column gutter
The area at the top of worksheets and crosstabs.

To add fields to a worksheet or crosstab, drag them to the column gutter.

comparison expression

An expression that compares two values and evaluates to Yes or No. For example:

Total <= 100 evaluates to Yes if the value in the Total field of the current record is less than or equal to 100; to No if the value is greater than 100.

Comparison expressions are useful when doing finds, creating calculated fields, and writing formulas used to validate field data.

Operators used in comparison expressions:

<	Less than
<=	Less than or equal to
=	Equal
<>	Not equal
>	Greater than
>=	Greater than or equal to

compound document

A document that contains a linked or embedded OLE object.

For example, if you have an OLE object from another application in an Approach file, the Approach file is a compound document.

constant
A value in a formula that is used exactly as you type it; it does not change from one record to another.

Constants can be numbers, dates, text, times, or Boolean values. Text, dates, and times must be enclosed in single quotation marks, like this:

'Price'

'1/1/95'

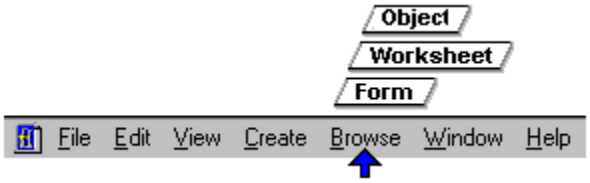
'11:30:00'

container application

An application that contains an OLE object.

For example, if you have an OLE object from another application in an Approach file, Approach is the container application.

context menu
The context menu always appears between Create and Window on the menu bar.

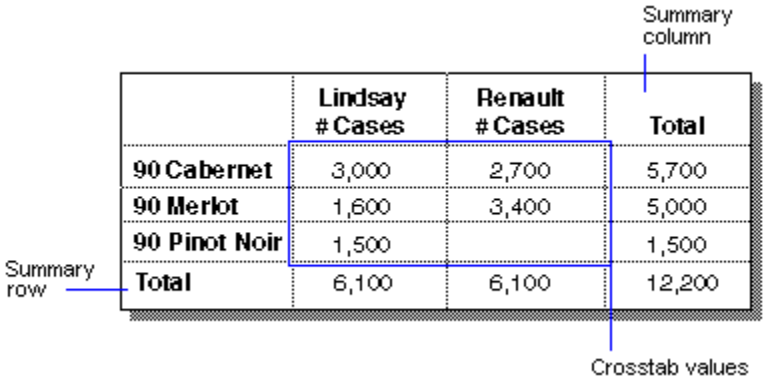


The context menu changes depending on the selected object or view. It provides commands appropriate for the selected object or view.

crosstab
Also called a cross-tabulation worksheet: A view that categorizes and summarizes data from many records. For example, here's a worksheet in a sales database. Notice that some records repeat the same data in the same field:

Sales Rep	Product	# Cases
Lindsay	90 Cabernet	3,000
Lindsay	90 Pinot Noir	1,500
Lindsay	90 Merlot	1,600
Renault	90 Cabernet	2,700
Renault	90 Merlot	3,400

A crosstab lets you collapse the five sales records into two summaries, one for each sales rep:



current record

The active record in a view. In Browse, you can enter and edit data in the current record.

In a form, the record showing is the current record.

In a worksheet, columnar report, or mailing label, the current record is the record you click. If no record is selected, the current record is the first record.

As you switch between views based on the same database, the current record stays the same.

database

A collection of data organized into fields and records.

An Approach file (.APR) is not itself a database, but it is associated with at least one database. When you choose File - New Database, you create (1) a database, in which you store data (for example, names, addresses, and so on), and (2) an Approach file, in which you create views that let you work with the data.

When you enter data, Approach immediately saves the data in the database.

You can join databases so that more than one database is associated with a single Approach file.

Approach works with many database file types, but you can work in Approach without having another database product on your computer.

Other database products use the terms "table" or "data file" for a collection of data organized into fields and records; unlike Approach, they reserve the term "database" for a collection of tables.

database password

A sequence of characters you must enter before you can access the data in a database.

The password grants you read-only or read-write access to the database(s) associated with an Approach file.

To define a password for a dBASE or FoxPro database, use File - User Setup - Approach Preferences and click the Password tab.

data
The information entered in a field within a record and stored in a database. Typically, data is text, numbers, dates, or times.

Approach automatically saves data as you enter and edit records.

date field

A field that can hold a single date. You can perform finds, sorts, and calculations on dates in a date field.

To format a date, in Design, double-click the field and use the InfoBox.

Regardless of the format of the date field, in Browse, you must enter a date month first, then day, then year.

default style
A collection of style and layout information that Approach uses to create new views, unless you select a SmartMaster template or application.

To modify the default style, choose File - User Setup - Approach Preferences and choose Default Style in the Display tab.

delimited text file
A text file that uses separators such as commas, spaces, or tabs to break up the text into discrete units. One row in a delimited text file represents one record.

When you open a delimited text file as a database in Approach, the units of text become data in fields.

descending order
The sort order that arranges records from

- Z to A, for text (usually case-insensitive)
- Largest to smallest, for numbers
- Latest to earliest, for dates and times

Design

The environment in Approach for laying out and designing views. In Design, you do tasks such as adding fields to views, adding color, changing label text, changing fonts, and creating text blocks.

To go to Design, do one of the following:

- Click the Design button in the action bar.
- Click the Environment button in the status bar and select Design.
- From the View menu, choose Design.
- Press CTRL+D.

You must save the work you do in Design by choosing Save Approach File from the File menu.

detail database

When an Approach file is associated with joined databases, each view not only can display the data of its main database; it can also display supplementary information from the other databases, giving details about the records of the view's main database.

When a database provides supplementary information, it's called a detail database.

If the main and detail databases have a one-to-many relationship, a repeating panel is a good way to display data from the detail database. The panel displays the "many" details of "one" record of the main database

A view can have only one main database, but it can have many detail databases.

drop-down box

A data-entry type for fields. In Browse, it offers a predefined list of values. Because it limits the values users can select and doesn't allow users to enter new values, this kind of field can help prevent the introduction of inaccurate data into your database.

To make a field a drop-down box: In Design, double-click the field to display the InfoBox. In the Basics tab, select Drop-down list as the data-entry type.

embed
To insert an OLE object in Approach. An embedded object gives you access to a server application when you are working inside Approach. When you embed an object

- You edit the embedded object by opening the server application from within Approach.
- The data is stored with the database associated with the Approach file, not in the server application.
- There is no link to the server application.

When you embed an object

- In Browse, in a PicturePlus field, it becomes part of a record.
- In Design, it becomes part of the background of a view.

expression
A combination of operators, operands, and functions that yields a single result. In Approach, you can create arithmetic, comparison, and logical expressions.

A formula can consist of one or more expressions.

field box

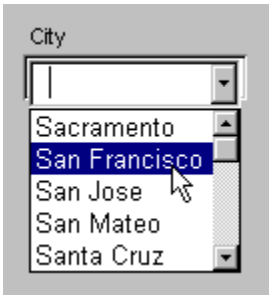
A data-entry type for fields. In Browse, type in the field box to enter data in the field.

The field box is the default data-entry type, so when you add a field to a view, Approach adds it as a field box. To change to another data-entry type: In Design, double-click the field to display the InfoBox. In the Basics tab, select another data-entry type.

field box and list

A data-entry type for fields. In Browse, it offers a predefined list of values and a field box that lets users enter a new value. The new value becomes part of the list.

To make a field a field box and list: In Design, double-click the field to display the InfoBox. In the Basics tab, select Field box & list as the data-entry type.



field

The smallest unit of data in a database. It's a good idea to break your information into as many meaningful fields as you can, especially if you plan to do finds on the data in the field.

For example, in an address database, rather than have just two fields for name and address, break the information into at least six separate fields: first name, last name, street, city, state, and postal code. You can then do finds for all persons living in the same city, all persons with the same last name using the same postal code, and so on.

- To define a field for a database: From the Create menu, choose Field Definition.
- After you define a field, you can add it to a view: In Design, from the context menu, choose Add Field.

field definition
A set of attributes that includes the field name, the type of data the field can contain, a maximum field length for some field types, and optional settings for controlling and validating data as it's entered in the field.

Every field in a database has a definition. Choose Create - Field Definition to define a field.

field mapping
Approach uses field mapping to ensure the accuracy of field definitions or the relationship between fields and data. You may need to map fields when you

- Open an Approach file in which previous changes to field definitions have not been saved in the Approach file.
- Import data from one database into another. Field mapping defines which fields should receive the data being imported.
- Import one Approach file (.APR) into another. Field mapping defines what kind of data, stored in fields that already exist in the receiving file, should appear in the fields of the views being imported.

field name
A name for a field, stored in the database as part of the field definition.

Follow these guidelines when you name a field: Begin the name with a character; use only letters, numbers, and underscores; use no more than 10 characters; avoid spaces and characters like $, &, @, and so on.

A field name is different from the label that identifies a field in a view:

Approach uses the field name as the label when you add the field. You can, however, change the label without changing the field name, and vice versa.

field reference

In formulas, a reference to a field. When calculating the formula, Approach uses the value in the referenced field from the current record.

For example, the calculated field Commission contains the formula

INVOICE.Amount * 0.05

The name INVOICE.Amount is a reference to the value in the Amount field of the INVOICE database. In one record, Amount is 3500; in another, 2800; so the formula result varies according to the value in the referenced field.

You must add the name of the database to the field reference when the Approach file has joined databases. The name of the database must be all uppercase letters, separated by a period (no spaces) from the field name.

field type
Also called "data type," the specification for the type of data you can enter in a field. Assign a field type to a field in the Field Definition dialog box.

Possible field types: Boolean, calculated, date, memo, numeric, PicturePlus, text, time, variable.

file type
The specification for the way a program stores and organizes data in files. In Approach, you can use a variety of database file types.

find condition

An instruction you give to Approach to find records in a database. A find consists of one or more find conditions.

A find condition can range from the simple, such as a single value you want to find, to the advanced, such as find conditions that use operators (=, <>, >, <, and so on) or formulas or If statements. You can also combine find conditions in a single find when you link the conditions with AND or OR.

The following is an example of a simple find condition expressed in English language format:

In database "Offices," find all records in field "City" that are exactly equal to "Tokyo"

When Approach finds records that match the find conditions, it displays only those records as the found set.

find

Also known as a search or query: To search for a set of records that satisfy one or more conditions you specify about data in one or more fields. For example, you might want to find all records in which the field Country contains France.

In Browse, click Find in the action bar to start a find.

Approach then gives you two ways to do a find:

- Find request. Use the blank view to enter your find conditions.
- Find Assistant. The assistant is especially useful for finding duplicate records, distinct records, and top or lowest values in a field.

The set of records that is the result of a find is called the found set. After a find, Approach keeps the found set as the only data you work with until you show all the records in the database or do another find.

If you know that you'll do the same find often, name the find so you can use it again.

Find (environment)
The environment in Approach in which you specify find conditions. In Browse, click Find in the action bar to go to the Find environment.

Then use the buttons that now appear in the action bar to create and execute the find.
When Approach finds records that match the find conditions, it returns you to Browse and displays only those records as the found set.

find request

One of the two ways Approach gives you to do finds. (The other is the Find Assistant.)

A find request is a blank view that you use for entering find conditions.

To create a find request, in Browse, click Find in the action bar. Approach then displays the tools for doing a find using a find request: a blank view in which to enter find conditions; buttons in the action bar to create, execute, and cancel the find; and a set of SmartIcons representing the operators you can use in the find conditions.

fixed-length text file
A text file in which the text is broken into blocks of a specific length. If you open a fixed-length text file as a database in Approach, the blocks of text become data in fields.

form
A kind of view that focuses on a single record. You can use the same form to see every record in the database, but the form shows you only one record at a time.

Forms are useful for entering and editing data for a record.

Forms can have up to five pages. They can also be converted into dialog boxes for use in macros or scripts.

To create a form: From the Create menu, choose Form.

form letter

A view that combines text you type with names and addresses from a database record, so you can send copies of the same letter addressed to many different people. You can also create envelopes printed with addresses for the form letters.

To create a form letter, choose Create - Form Letter.

found set

The group of records that match your find conditions.

After a find, Approach keeps the found set as the only data you work with until you show all the records in the database or do another find.

full record locking

A method of network data-sharing in which only one user at a time can edit a record, although more than one person can look at a record at the same time; the opposite of optimistic record locking.

To turn on full record locking, choose File - User Setup - Approach Preferences. In the General tab, deselect optimistic record locking.

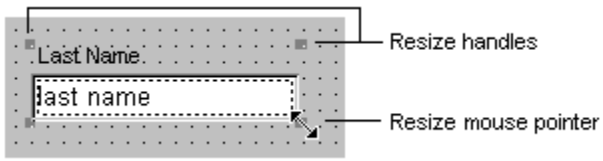
function

A built-in formula that performs a specialized calculation automatically, often by using values that you supply. Such values are called parameters.

Some functions perform simple calculations. Many, however, simplify your work by performing complex calculations; for example, SNPV calculates the net present value of an investment based on a series of periodic cash flows and a discount rate.

grid
A non-printing matrix of dotted lines that you can show in Design. The grid provides a background to help you lay out objects in a view.

handles
In Design, squares at the edges of a selected object. To resize the object, drag one of its handles.



If you are enlarging a field that is in a report panel or repeating panel, be sure the resized field stays completely inside the panel.

index

A file that contains a list of the values of a field, with pointers to the records in the database that contain those values. Finds and sorts on a field go faster when the field is indexed.

Approach creates an index for a field (excluding memo fields) the first time you do a find or sort on that field. It maintains the index as you enter data.

Approach index files have the file extension .ADX. Each .APR can have only one .ADX file associated with it. Approach stores indexes for all the fields in the single .ADX file.

If you delete the .ADX file, Approach creates a new one with the next sort or find that you do. When you compress a file, Approach rebuilds the index file.

InfoBox

A window that shows the properties of the selected object, such as its name, color, borders, alignment, and so on.

The InfoBox can display the properties of every object in a view, including the view itself. To display the InfoBox so you can change those properties, go to Design and then do one of the following:

- Double-click the object whose properties you want to change. For a view, double-click the background, away from any other object.
- From the context menu, choose the Properties command of the object.
- Select the object and click the InfoBox icon:

You can keep the InfoBox open as you work in Design.

The InfoBox displays the object's properties on one or more tabs. Click the tabs to see the properties.

Changes you make to a property happen either at once or as soon as you click elsewhere, so you can see the effects of your changes immediately and decide whether you like the results.

join

To link two databases so that you can work with the data of all the databases from inside a single Approach file; show relationships between the data of the joined databases; do finds; and so on.

You create a join by linking one or more fields that the databases have in common.

For example, the following illustration shows a diagram of two joined databases, one with data about authors, the other with data about publishers. They're joined on the Publisher and Name fields because those fields contain data that both databases have in common.

bmc join.bmp

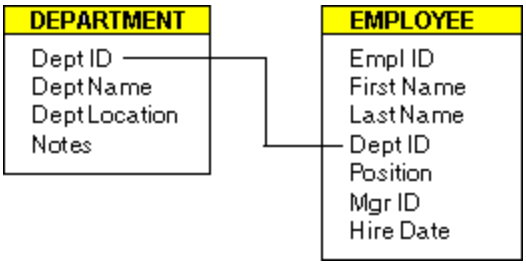
Joined databases are a fundamental part of a relational database application like Approach. They reduce the amount of data entry you have to do and increase the power and ease with which you can manipulate your data.

To join databases: From the Create menu, choose Join.

join field

A field in one database linked to a field in another database, thus joining the databases. Join fields don't have to have the same name, nor do they have to be defined as the same data type; but they must contain the same kind of data.

Often, a join field is a numeric ID field created specifically for joining. In the following illustration, the employee database contains personal data, such as first and last names, address and phone number. The department database contains department information, such as manager, location, and cost center. The two databases are joined on the Dept ID field because that is how the two databases are related: each employee is assigned to a department, and every department comprises employees.



key field

One or more fields that can identify each record in a database as unique, for example, an invoice number or an ID number.

When exporting data from Approach or saving a database file to a different file format, you must specify one or more key fields. If you do not specify a key field that is unique, an error message displays indicating that you can't export or save the file because a duplicate key exists.

label

Text used to identify a field in a view, stored in the Approach file (.APR) as part of the design of a view.

Approach uses the field name as the label when you add the field to the view. You can, however, change the label without changing the field name, and vice versa.

link
To place a copy of an OLE object in Approach, with a connection to the original object in the server application. If the original object changes, the copy also changes in Approach.

When you link an object

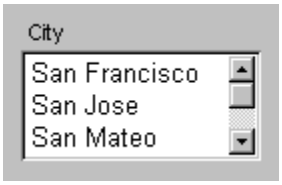
- In Browse, in a PicturePlus field, it becomes part of a record.
- In Design, it becomes part of the background of a view.

list box

A data-entry type for fields. In Browse, it offers a predefined list of values. Because it limits the values users can select and doesn't allow users to enter new values, this kind of field can help prevent the introduction of inaccurate data into your database.

Unlike a drop-down box or field box and list, however, you can size the list box to display as many of its values as you like.

To make a field a list box: In Design, double-click the field to display the InfoBox. In the Basics tab, select List box as the data-entry type.



logical expression
An expression that evaluates one or more comparison expressions and returns a result of Yes or No.

For example, suppose the field Orders contains 462 and the field ShipDate contains the date 1/15/95:

- (Orders >= 500) AND (ShipDate >= '1/1/95') returns No
- NOT (Orders >= 500) returns Yes

Operators used in logical expressions:

AND	Returns Yes if both expressions are true
OR	Returns Yes if either or both are true
NOT	Evaluates a comparison expression and returns Yes if it's false, No if it's true

lookup
An automatic display of data from a many-to-one or one-to-one relationship.

macro button
A control that you can add to a form, report, or other view. When you click the button, it executes an attached macro or script.

To create a button, choose Create - Control - Button.

Attach a macro to the button in the Macros tab of the InfoBox for the button.

macro
A single command that executes a sequence of other commands. You define this sequence when you create the macro.

Use macros to automate Approach tasks.

To create a macro, choose Edit - Macros - New.

mailing label
A view that displays database fields and text in a mailing address format. You can then print the addresses on standard or custom mailing labels.

main database

Every view in an Approach file represents a database. The database represented by the view is called its main database. The main database determines how many, and what kind of, records can be visible in the view.

When an Approach file is associated with joined databases, each view can have a different main database. A view must have one main database, and it can have many detail databases.

To find out what the main database of a view is, in Design, select the view and then look in the Basics tab of the InfoBox.

Example: Suppose a company has two joined databases, DEPT (departments) and EMP (employees). One view has DEPT as its main database. The company has five departments, so that view can show a maximum of five records.

The company has 100 employees. A view that has EMP as its main database can show a maximum of 100 records.

many-to-many
A relationship in which two or more records in one database can be related to two or more records in a joined database. For example, each order can include several products, and each product can appear on several orders.

In Approach, you can create a many-to-many relationship between databases, but it is better to design your databases so that you can create one-to-many or many-to-one relationships.

many-to-one
A relationship in which two or more records in one database can be related to only one record in a joined database. For example, several employees can be in the same department.

memo field

A field that can hold large amounts of text (maximum size: 64K). Define a field as memo instead of text if the field has to store more than approximately 254 characters, for example, customer profiles or specifications of products.

Design your database so that you can avoid doing finds or sorts on memo fields.

Data of memo fields is not stored in the database file itself; rather, it is stored in a file with the extension .DBT (for dBASE), .DBQ (for Paradox 3.5), or .MB (for Paradox 4.0).

Another way to store a large document as part of a record: Define a PicturePlus field and embed the document in it as an OLE object.

named style
A set of layout and style properties, such as font and color, that you name and save as a group. You can then apply the set of properties again, as a group, rather than applying individual properties one at a time.

You can apply a named style to a particular object or to a view as a whole; Approach uses only the properties that pertain to the selected object.

To create or apply named styles, in Design, double-click the view or object and click the Named Style tab in the InfoBox.

numeric field
A field that can hold numbers and can be formatted with numeric symbols, such as a currency symbol. You can perform arithmetic calculations on data in a numeric field.

To format a number, in Design, double-click the field and use the InfoBox.

OLE object

An object you embed or link in Approach through OLE. The object can be

- Part of the layout of a view. In Design, choose Create - Object.
- Data in a PicturePlus field, if you define the field to accept an OLE object. In Browse, select the field to activate the OLE server application.

Double-click an OLE object to edit it using the tools from the server application without leaving Approach.

Approach also lets you use OLE in the other direction--that is, you can embed an Approach OLE object inside another application.

OLE (Object Linking & Embedding)
A method for data exchange, using links, and for the creation of compound documents, using embedded objects, that lets you use one application from inside another.

For example, in Approach, you can use OLE to

- Add data to a record by embedding a Lotus WordPro document in a PicturePlus field.
- Use a SmartMaster background from Lotus Freelance Graphics as the background of an Approach view.

Approach also lets you use OLE in the other direction--that is, you can embed an Approach OLE object inside another application.

one-to-many
A relationship in which a record in one database can be related to two or more records in a joined database. For example, one department can have several employees. In a form, you use a repeating panel to represent the "many" data from a one-to-many relationship.

one-to-one
A relationship in which a record in one database is related to only one record in a joined database. For example, a vehicle number can be related to a license number for a single vehicle.

operand
A value in a formula, used in conjunction with one or more operators. Operands can be either constants (numbers, dates, text, or times) or field references. In the following formula, INVOICE.Amount (a field reference) and 0.05 are operands; * is the operator:

INVOICE.Amount * 0.05

operator
A symbol in a formula that defines the calculation or other evaluation to be performed. A plus sign (+) and a less-than sign (<) are examples of operators.

optimistic record locking

A method of network data-sharing in which two users can edit a record at the same time; the opposite of full (or pessimistic) record locking.

When the second user tries to enter changes, Approach warns that the second user's changes will write over those of the first user.

Approach uses optimistic record locking unless you deselect this option in File - User Setup - Approach Preferences, the General tab.

Approach runs faster with optimistic record locking on because it does not have to check whether to lock a record when a user tries to view it. Use this method when no more than one user at a time is likely to try to edit a record.

.

panel
A report component, visible in Design, that determines how field data appears in the report.

parameter
A value to be operated on in a function. Parameters appear in parentheses after the function name and can be either constants or field references. For example, the function Fill(text,number) could take the parameters

Fill('Baden',2)

to return BadenBaden.

PicturePlus field

A field that can contain graphics or OLE objects.

To allow freehand drawing with the mouse on top of a graphic or OLE object in a PicturePlus field: go to Design, double-click the field to display its InfoBox, and then on the Basics tab, select Allow drawing.

PowerClick reporting

An Approach feature that allows you to modify an existing columnar report. Use PowerClick reporting to reorganize and summarize data.

In Design, be sure View - Show Data is turned on.

Select the field to use for grouping records. Then, from the Column menu, choose Groups & Totals. Then choose a leading or trailing summary option. You can also use these PowerClick icons:

Next, select the column to summarize. From the Column menu, choose Groups & Totals and choose a calculation option. You can also use these PowerClick icons:

Sum (grand total):

Maximum:

Standard deviation:

Count:

Minimum:

Variance:

Average:

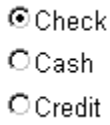
Print Preview
The environment in Approach that shows views on the screen as they appear when printed.

radio button

A data-entry type for a field that accepts only one of a small number of predefined values.

In Browse, to enter data in the field, you must select one of the values offered.

To make a field a set of radio buttons: In Design, double-click the field to display the InfoBox. In the Basics tab, select Radio buttons as the data-entry type. Define a clicked value and a label for each button.



read-only access
Permission to read data, but not to modify it.

- Make individual fields read-only. In Design, double-click the field to bring up the InfoBox and click the Basics tab.
- Assign a database password to your files to grant read-only permission to users who know the password. Choose File - User Setup - Approach Preferences and click the Password tab.

read-write access
Permission to read and modify data in a database. You can assign a database password to your files to grant read-write permission to users who know the password.

Assign a database password to your files to grant read-write permission to users who know the password. Choose File - User Setup - Approach Preferences and click the Password tab.

record
One set of related information in a database. For example, in an employee database, the information on each employee (name, address, date of birth, and so on) is a record.

relational database application

A database application that lets you bring together data from more than one database in a single form, report, or other view. Approach is a relational database application.

The distribution of data among several databases, which you then join, is a basic concept of a relational database application such as Approach. If you create multiple databases and join them rather than try to put all your data into a single database, you save yourself work and increase the variety of ways you can view and manipulate your data.

Plan your databases and store your information so as to take advantage of the power and efficiency of joined databases.

repeating panel

In an Approach file that has joined databases, a repeating panel is an object you add to a form to display the "many" side of a one-to-many relationship between the main database of the form and one of its detail databases.

The repeating panel shows data from multiple records in a detail database that are related to the current record of the main database.

For example, a repeating panel in a department form might list all the employees in the department.

To create a repeating panel: In Design, from the Create menu, choose Repeating Panel.

report
A view used for organizing, summarizing, and presenting data from many records. A report shows all the records in the database or the current found set on one or more pages.

row gutter
The area at the left side of worksheets and crosstabs that holds row headers.

To convert a worksheet to a crosstab, drag a column header from a worksheet and drop it in the row gutter.

server application
An application used to create an OLE object.

server
One or more central computers that store files and applications to which users have access across a network.

SmartIcons

Buttons (icons) in the Approach window that let mouse users choose commonly used commands and macros.

To select SmartIcons, click them.

To see a short description of what an icon does, leave the mouse pointer on the icon for a few seconds.

To change the set of SmartIcons, choose File - User Setup - SmartIcons Setup.

SmartMaster application

An Approach file with one or more associated databases, designed to be a ready-to-use application for business or personal use.

To create an application based on a SmartMaster application, choose File - New and select a SmartMaster application.

To create a SmartMaster application, save an Approach file as an .MPR file to the SmartMaster directory. Define the directory in the General tab of the Approach Preferences dialog box.

SmartMaster template

A predefined set of field definitions for a single database file. Approach provides SmartMaster templates for many common business and personal uses, such as a customer database and an employee database.

To create a new database from a template rather than defining all of your fields from scratch, choose File - New and select a SmartMaster template.

To start from the basis of an even more fully developed sample business application, select a SmartMaster application.

sort

To organize records alphabetically, numerically, or chronologically by data in one or more fields. You also designate whether to organize the records in ascending (for example, A - Z) or descending order (Z - A).

You can sort an entire database or a found set.

sort field

A text, numeric, date, or time field used for sorting records in a database.

The first field you select is the primary sort field. Subsequent fields you select resolve conflicts in the sort order that arise when the same data appears in the primary sort field in different records.

For example, when sorting a database of names, select Last Name as the primary sort field and First Name as the second sort field so that Smith, John and Smith, Alice appear in correct alphabetical order.

status bar

The bar at the bottom of the Approach work area.

The information available from the status bar changes depending on the current environment. You can always, however, click the buttons on the far right to change to another view or environment.

In Browse, the status bar indicates whether you are working with all records of a database or a found set.

summary function
A function that applies to a group of records. For example, the SSum(Amount) summary function adds the values in the Amount field in a range of records you specify.

summary panel
An area in a report that may contain a calculated field that summarizes data.

summary report
A report that omits record-by-record detail and displays only summary information.

tab order

The order in which you move through the fields in a view when you press TAB. Also referred to as data-entry order.

To change the tab order, in Design, choose View - Show Tab Order and then change the numbers that appear. Use the buttons in the action bar to confirm your changes.

Calculated fields cannot be included in the tab order.

text field
A field that can hold any characters you can type, including letters, numbers, and symbols. The maximum length of a text field is approximately 250 characters. If you need to hold larger amounts of text in a field, define the field as memo.

You can search on a text field using any character in the field.

time field

A field that can hold a single time. You can perform finds, sorts, and calculations on times in a time field. To format a time, in Design, double-click the field and use the InfoBox.

Tools palette

In Design, a set of buttons for drawing shapes and lines, creating text blocks, and adding fields.

To show the Tools palette, choose View - Show Tools palette.

To move the palette around in the work area, drag its title bar.

variable field

A field that temporarily stores a value, which can be text, a number, a date or time, or a Boolean value. When you define a variable field, set its data type and initial value. The value is the same for every record in your database application.

Use variable fields to store intermediate values in calculations and macros, and to pass values between Approach and Notes using Notes/FX.

Variable fields are stored in the Approach file (.APR).

view

An interface that you create in an Approach file (.APR). Views let you look at and work with the data stored in the file's associated database files. Views are stored in the Approach file.

When you create a new file, Approach provides a standard form and a worksheet. You can modify these views and add as many other views as you need.

You can design each view. In Design, double-click the background and the objects to display the properties of objects, including the view itself, in the InfoBox. To save design work: From the File menu, choose Save Approach File.

Approach provides these kinds of views: form (for looking at one record at a time), report, form letter, mailing labels, worksheet, crosstab, chart, and envelope.

view tab
The folder tab that appears at the top of the window for each view in the Approach file.

Click the tab to go to that view.

Go to Design to

- Double-click the tab to change the name. (Maximum number of characters: 79.)
- Drag the tab to move the view to another position.

worksheet

A view that presents data from many records in a grid of columns and rows. Each record occupies a single row in the worksheet.

A worksheet shows all the records in the database or the current found set.

zoom
To change the magnification of a view on the screen. You can zoom in for a closer look or zoom out for the big picture. Zooming does not affect the size of a view when you print it, only how it appears on the screen.

How do I access data from a database using a batch process?

There are two strategies for working with data using LotusScript in Approach:

- You can access values in records through the fields in a view
- You can bypass the user interface and access data directly from a database

LotusScript lets you automate the first type of data access by controlling elements of the Approach interface that you are already familiar with: placing fields in a view and modifying the value entered in the field.

The second strategy accesses the data in a fast batch process without using the Approach user interface. This data access process involves the following operations:

- Establishing a link to a database using a Connection object
- Selecting the records to access from the database using a Query object
- Storing the required information locally while it is in use using a ResultSet object

To understand more about when to use this method for accessing data, see Connection class.

The following example illustrates each of the steps for accessing data directly. It uses these objects:

- Document (referred to using the global product variable CurrentDocument)
- Connection
- Query
- ResultSet

Example - accessing database data directly

The following script comes from the Approach Meeting Room Scheduler SmartMaster application. To open this SmartMaster, choose File - New Database and select the Meeting Room Scheduler. The script is a global sub.

```
Sub ReadBlock(DateReserved As String)
   ' DateReserved      Date formatted as a string
   ' Declare objects for connecting to the reservation
   ' database.
   Dim C As New Connection
   Dim Q As New Query
   Dim RS As New ResultSet
   Dim s As Double       ' Start time of existing reservation
   Dim f As Double       ' End time of existing reservation
   Dim Row As String     ' Room reserved
   Dim n As String       ' Reservation owner
   Dim Tname As String   ' A shorter reservation table name
                         ' reference
   Dim FullTname As String   ' Table name and path reference
   ' Collect the name of the first table associated with the
   ' document, numbered starting at zero.
   Tname = CurrentDocument.Tables(0).TableName
   ' Place the names of the current rooms in the view.
   Call DisplayRooms()
   ' Build the connection to retrieve the reservation
   ' information for the passed-in date. This is a standard
   ' data-access sequence. To reuse it, modify SQL SELECT
   ' statement as needed.
   ' Note that the database is dBASE IV in this case.
   If C.ConnectTo("dBASE IV") Then
      Set Q.Connection = C
      ' Table name for the query needs to have full path.
```

```
      FullTname =  CurrentDocument.Tables(0).Path & Tname & _
         ".dbf"
      ' The query is set to retrieve values from all the table
      ' fields for records whose Date Reserved field matches
      ' the date.
      ' Note that the syntax for the SQL SELECT statement
      ' requires extra quotation marks to define the string
      ' concatenation.
      Q.SQL = "SELECT * FROM """+FullTname+""""+ Tname+ _
         " WHERE (("_
         + Tname+"".""Date Reserved"" = ' "+DateReserved+"' ))"
      ' Assign this query to the result set.
      Set RS.Query = Q
      ' Use the result set to fill in the reservation
      ' information in the view.
      ' If the result set was created successfully, then...
      If (RS.Execute) Then
         ' Confirm that there are reservations for this date.
         If (RS.NumRows) Then
            RS.FirstRow    ' Go to the first record in the
                           ' result set.
            ' Loop through all of the records in the result set
            ' and display the reservation information in the
            ' view.
            Do
               s = RS.GetValue("Start Time")
               f = RS.GetValue("End Time")
               Row = RS.GetValue("Room Name/Number")
               n = RS.GetValue("Reserved By")
               ' Build a text block in the view with the
               ' reservation information from this pass through
               ' the loop by calling the global function
               ' DisplayBlock.
               Call DisplayBlock(n, s, f, row)
            Loop   While RS.NextRow
         End If      ' NumRows not zero
      End If      ' Result set successful
   End If      ' Connection successful
   ' Close the connection to allow other connections to this
   ' database.
   C.Disconnect
End Sub
```

How do access data from a Notes database?

The Connection, Query, and ResultSet objects let you access Notes data from Approach. You might want to do this to create Approach reports or to search for data in a Notes database.

To create a connection to a Notes database, you must know the following:

- The data-source type of the database: Is the database on a server, on your local hard disk, or on the workspace?
- The name of the table to search.
- The user name and password, if required to access the table.
- The fields that you want to use in the find.

Connecting to a Notes database on a server involves the following operations:

- Determining the exact server name. To do so, choose File - Open; in "Files of type," select Lotus Notes - Server; and note the name of the server you are working with. The following is an example of a server name:

  `CN=Approach_OU=SJC_OU=A_O=Lotus`
- Changing the underscores (_) in the server name to forward slashes (/). The example server name becomes the following:

  `CN=Approach/OU=SJC/OU=A/O=Lotus`
- Determining the database name, including path. An example database name is EXAMPLES\BUSCARD.NSF.
- Determining the user name and password, if necessary to access the database name.
- Building ConnectTo method arguments. The following table describes these arguments.

ConnectTo method argument	Value
DataSourceType	Lotus Notes - Server
UserId	A string
Password	A string
Database	ServerName!DatabaseName

Example - connecting to a Notes database on a server

The following connection script shows how these pieces come together in the ConnectTo method statement:

```
Dim Con As Connection
Dim Qry As Query
' Make the connection.
' No UserId or Password is required to access this database.
If Con.ConnectTo("Lotus Notes - Server",,,_
   "CN=Approach/OU=SJC/OU=A/O=Lotus!Examples\Buscard.nsf") then
   Set Qry.Connection = Con
   ' Continue building query and result set.
   Con.Disconnect
End If
```

Example - connecting to a local Notes database

The following example illustrates a connection to a local Notes database. The text of this script is stored in C:\ <*PATH*>\Samples\dw08_s1.lss. It does the following:

- Determines the types of connections you can make and prints a list to the Output panel of the IDE.
- Makes a "Lotus Notes - Local" data source connection and connects to a database called NAMES.NSF.
- Lists all of the tables in NAMES.NSF and opens one called People.
- Searches that table in the First Name and Last Name fields and prints the names of the people in that table to the IDE.

```
Sub Click(Source As Button, X As Long, Y As Long, _
   Flags As Long)
' Declare objects for a Connection, Query, and ResultSet.
   Dim MyConnection As New Connection
```

```
   Dim MyQuery As New Query
   Dim MyResultSet As New ResultSet
' Declare a set of variables for an array, and for the
' arguments of the above objects.
Dim MyArray As Variant            ' Array for temporary
                                  ' storage
Dim MyPath As String              ' Data source path
Dim MyDatabaseSource As String    ' Current data source type
Dim MyDatabase As String          ' Database name
Dim MyTable As String             ' Table name
Dim MyUserId As String            ' User login name
Dim MyPassword As String          ' User password
Dim i As Integer                  ' MyArray index
Dim LastName As String            ' Data retrieved from table
Dim FirstName As String           ' Data retrieved from table
' Print a list of data source types available in the output
' panel of the IDE. Use this list to determine the exact string
' required (in the ConnectTo method) to make the connection to ' the data source.
MyArray = MyConnection.ListDataSources()
' Use LotusScript functions LBound (lower bound) and Ubound
' (upper bound) to determine the limits of the data source
' types array.
For i = LBound(MyArray) To UBound(MyArray)
   ' Print the data source types to the Output panel of the
   ' IDE.
   Print "Data source("i") = "MyArray(i)
Next
' Set the arguments for the connection object.
' Tip: You could use an input box to get this information from
' the user and connect to whatever the user specifies.
MyDatabaseSource = "Lotus Notes - Local"
' Change this path to match your operating system.
MyPath = "C:\NOTES\DATA\"
MyDatabase = "NAMES.NSF"
' This table has no user or password requirements, so these
' strings are blank.
MyUserId = ""
MyPassword = ""
' Connect to the database.
If MyConnection.ConnectTo(MyDatabaseSource, MyUserId, _
   MyPassword, MyPath & MyDatabase) Then
   ' List the available tables in the database.
   MyArray = MyConnection.ListTables()
   For i = LBound(MyArray) To UBound(MyArray)
      ' Print the data source types to the IDE Output panel.
      Print "Table("i") = "MyArray(i)
   Next
   ' Specify the table that data will be extracted from.
```

```
   MyTable = "People"
   ' Set the Connection property of the Query object to
   ' the current connection.
   Set MyQuery.Connection = MyConnection
   ' Specify the table to be searched, including path.
   ' The ampersand (&) concatenates pieces of the string.
   MyQuery.TableName = MyPath & MyDatabase & "\" & MyTable
   ' Set the Query property of the ResultSet to the query.
   Set MyResultSet.Query = MyQuery
   ' Get the data from the table and put it in the result set.   If
(MyResultSet.Execute) Then
      ' Make sure there is data in the table.
      If (MyResultSet.NumRows) Then
         ' Start at the first row.
         MyResultSet.FirstRow
         ' Loop through the records and get the values from
         ' the First_Name and Last_Name fields.
         Do
            LastName = MyResultSet.GetValue("Last_Name")
            FirstName = MyResultSet.GetValue("First_Name")
            ' Print the data to the Output panel of the IDE.
            Print "Person" & MyResultSet.CurrentRow & _
               "in the table is " & FirstName & " " & LastName
         ' Continue looping until there are no more rows.
         Loop   While MyResultSet.NextRow
      Else
         ' If the table is empty, warn the user.
         Messagebox "The table is empty.", MB_OK + _
            MB_ICONINFORMATION + MB_APPLMODAL, "Empty Table"
      End If   ' If there is data in the table.
   Else
      ' If the result set was not successful, warn the user.
      Messagebox "The query did not succeed. Check the " & _
         "connection.", MB_OK + MB_ICONINFORMATION + _
         MB_APPLMODAL, "Unsuccessful Query"
   End If   ' If the result set was successful.
Call MyConnection.Disconnect()
Else
   ' If the connection was not successful, warn the user.
   Messagebox "Connection failed", MB_OK + _
      MB_ICONINFORMATION + MB_APPLMODAL, "Connection"
End If   ' If the connection was successful.
End Sub
```

How do I change the summaries in a report?

This example uses the same report to display more than one type of summary information, according to user input. Changing the summaries in an existing report involves the following operations:

- Prompting the user for a field to group records by.
- Grouping the panel on the input field.
- Displaying the report in Print Preview mode.

The following script illustrates these operations.

Example - displaying a different summary report

This example is part of the application stored in C:\<PATH>\Samples\dw08_s6.APR.

When the user switches to the report, this script prompts for how to group the report summaries. The original report is columnar with summary groupings.

```
Sub SwitchTo(Source As Report, View As View)
' SwitchTo event for a report object
' * RUN-TIME DEPENDENCIES
' * Files: The report that the script is attached to is a
' * summary report with a group-by field.
   Dim MyGrp As String   ' Store user input.
   Dim Flag As Integer   ' Indicates successful field name
                         ' entry.
   ' Go to the Browse environment.
 CurrentApplication.ApplicationWindow.DoMenuCommand(IDM_browse)
   ' Prompt the user for the field name by which the user wants
   ' to group records.
   Flag = 1
   While Flag = 1
      MyGrp = InputBox$("What do you want to group by?_
         (Date, Product, RepID)"&Chr(10)&Chr(13)&"Date"&_
         Chr(10)&"Product", "Grouping", "Date", 300, 300)
      If MyGrp <>"Product" And MyGrp<>"RepID" And _
         MyGrp<>"Date" Then
         Flag = 1
         Beep
         Messagebox "Invalid Group Field." & Chr(13) & _
            "Try Again.",0+48+0+0, "Invalid Entry"
      Else
         Flag = 0
      End If
   Wend
   ' Change the current summary panels (leading and trailing)
   ' to reflect the new grouping.
   Source.Summary.GROUPBYDATAFIELD = MyGrp
   Source.Summary.MyLPanelFld.Datafield = MyGrp
   Source.Summary.MyLPanelFld.LabelText = MyGrp
   ' Go to Print Preview so that the user is prompted to sort
   ' on the new grouping and can see the results.
CurrentApplication.ApplicationWindow.DoMenuCommand(IDM_Preview)
End Sub
```

How do I control how users enter data?

LotusScript gives you control over the flow of your application by allowing you to set the keyboard focus, prompt for user input, determine which commands are available at a given time, and determine which actions happen by default.

The Meeting Room Scheduler SmartMaster application provides several examples of controlling the flow of an application. One sequence in particular executes the following operations:

- Switching to a form displayed as a dialog box when the user clicks a button.
- Setting the focus on the form to indicate where user input is required.
- Displaying the room reservations using the date entered.
- Duplicating the dialog box closure so that both clicking the mouse and pressing ENTER complete the action.

These operations are illustrated in following scripts. They are part of the Approach Meeting Room Scheduler SmartMaster application. To open this SmartMaster, choose File - New Database and select the Meeting Room Scheduler.

Example - clearing data in a dialog box

The first sub is attached to the SwitchTo event for the Enter Date form. This form is already set in the InfoBox to display as a dialog box.

```
Sub SwitchTo(Source As Form, View As VIEW)
' SwitchTo event for the Enter Date form
' This script clears any text from the field box for date entry
' on the form. The Enter Date form is set to display as a
' dialog box.
   Source.Body.fbxDate.Text = ""
End Sub
```

Example - button script to check for valid user input

The next sub processes input when the user clicks the OK button on the form.

```
Sub Click(Source As Button, X As Long, Y As Long, _
   Flags As Long)
' Click event for the btnOK Button object on the Enter Date
' form. This script processes the date entered in the fbxDate
' field box.
   ' Display the schedule on the Schedule Display view.
   Call ProcessDate()  ' Global sub that checks that the date
' is valid and displays the schedule for
                       ' that date.
End Sub      ' Click event for btnOK on the Enter Date form
```

Example - keyboard script to check for valid user input

The final script is almost the same as the previous one, except it runs when the user presses ENTER to close the dialog box instead of clicking OK. The sub is attached to the KeyDown event for the user entry text block on the Enter Date form. The ENTER key translates to a character code of 13.

```
Sub KeyDown(Source As Fieldbox, CharCode As Long, _
   Repeats As Integer, Flags As Integer, _
   OverrideDefault As Integer)
' KeyDown event for the fbxDate FieldBox object on the Enter
' Date form
' When the user presses ENTER while the focus is on this field ' box, this script
processes the date entered.
' If the KeyDown event returns an ENTER key (character code 13)
   If CharCode = 13  Then
      ' Display the schedule for that date.
```

```
      Call ProcessDate()  ' Global sub that checks that the
                          ' date is valid and displays the
                          ' schedule for that date.
   End If   ' If the ENTER key is used in fbxDate
End Sub     ' KeyDown event for fbxDate on the Enter Date form
```

How do I create a document to display the result set?

When you create a result set, there are two ways to show the retrieved data in an Approach view:

- Use the result set as a table associated with a new Approach document. Use this technique if you are going to use this subset of information for tasks that require user input, such as finding records with user input or building reports.
- Display data from the result set in text blocks or other display elements. Use this technique if you are using the information from the result set in addition to data already available in a view. For example, add text blocks containing result set information to a report that already contains information from another database.

  For more information, see How do I display data from a result set in a view .

Approach documents (.APR files) are instances of the Document object. To create a new document to display a result set you must do the following:

- Create a connection to an existing table.
- Create a general query to extract records from that table.
- Create a result set to contain extracted records from the table on disk.
- Create a new .APR file using this result set.

The following example illustrates each of these steps. The text of this script is stored in C:\<*PATH*>\Samples\dw08_s5.lss.

Example - displaying a result set in a new document

The CreateDocument sub creates a new document in Approach that displays the data from a result set. The result set is the main table for the new document.

```
Sub CreateDocument
' * RUN-TIME DEPENDENCIES
' * Files and paths: This script depends on an existing
' * dBASE IV database in the directory:
' * C:\LOTUS\WORK\APPROACH\BLANK.DBF
' Declare the necessary variables.
Dim MyConnection As New Connection   ' Connection to a table
Dim MyQuery As New Query             ' Query used to extract
                                     ' data from the table
Dim MyResultSet As New ResultSet     ' Result set to contain
                                     ' extracted data in the
                                     ' new document
Dim MyDoc As Document                ' New Document object
' Specify the type of database (dBASE IV) to connect to as the
' source for the new document.
If MyConnection.ConnectTo("dBASE IV") Then
   ' Specify the name of the connection MyConnection that the
   ' query MyQuery uses to create the new document.
   Set MyQuery.Connection = MyConnection
   ' Specify the full path and table name to be used as source
   ' for the new document.
   MyQuery.TableName = "C:\LOTUS\WORK\APPROACH\BLANK.DBF"
   ' Add a statement to specify a subset of records to be
   ' returned in the result set here. "MyQuery.SQL = ..."
   ' Specify the result set in the new database to receive the
   ' new records from the query.
   Set MyResultSet.Query = MyQuery
   ' If the connection and query succeed, create the new
   ' document.
```

```
   If (MyResultSet.Execute)Then
      Set MyDoc = New Document(MyResultSet)
   End If   ' If the result set was successful.
   ' Error handling for a failed result set would go here.
   ' Disconnect from the source table.
   MyConnection.Disconnect
End If   ' If the connection was successful.
' Error handling for a failed connection would go here.

End Sub
```

How do I display data from a result set in a view?

When you create a result set, there are two ways to show the retrieved data in an Approach view:

- Display data from the result set in text blocks or other display elements. Use this technique if you are using the information from the result set in addition to data already available in a view. For example, add text blocks containing result set information to a report that already contains information from another database.
- Use the result set as a table associated with a new Approach document. Use this technique if you are going to use this subset of information for tasks that require user input, such as finding records with user input or building reports.

  For more information, see How do I create a document to display the result set .

To display information from the result set in a view, create display elements and place them in a view. You can modify display elements using these properties:

- Background
- Border
- Color
- Height, Width
- Top, Left
- Name

The script that accomplishes this task involves the following operations:

- Creating the text block (or other display element) to hold the information.
- Filling the text block with the correct value from the result set.
- Setting the display properties for the text block so it matches the view.
- Positioning the text block in the view.
- Naming the text block.

The following script illustrates this process. It is part of the Approach Meeting Room Scheduler SmartMaster application. To open this SmartMaster, choose File - New Database and select the Meeting Room Scheduler.

Example - displaying found data in a view

The example sub, DisplayBlock, displays the owner of a room reservation in the correct time slot in a view that displays the reservation information for a particular day. The sub is called from another sub that creates a result set for the specified day and passes the reservation information to DisplayBlock.

```
Sub DisplayBlock(Txt As String, Start As Double, _
   Finish As Double, RoomName As String)
' Display the reservation owner in the correct time slot
' in the current view body.
' DisplayBlock is called from readBlock.
   '  Txt            reservation owner
   '  Start          reservation start time
   '  Finish         reservation end time
   '  RoomName       reserved room's name or number
' * RUN-TIME DEPENDENCIES
' * Constants: Uses constants defined by LotusScript defined in
' * LSCONST.LSS.
' * Globals: Uses the global array Rooms() filled by the
' * readBlock sub.
' Declare variables.
   Dim Tt As textbox     ' New text block to hold the
                         ' reservation owner's name in the view
   Dim i As Integer      ' Index of array with the room names
```

```
' Index of the room that matches the roomName passed in. It is
' used to determine the vertical placement of the reservation
' in the view.
   Dim MatchedRoom As Integer
' Offset and multiplier for the vertical placement of the
' reservation.
   Dim VerticalPlacement As Integer
' Search through the global array Rooms to find the room passed
' in from the schedule database using the global sub readBlock,
' also part of this .APR file.
   ' Set MatchedRoom to the index of the room passed in.
   For i = 0 To UBound(Rooms)
      If Rooms(i) = roomName Then
         MatchedRoom = i
         i = UBound(Rooms)
      End If   ' If element matches the room passed in.
   Next
' Set position and display for the reservation.
' Header in the view takes up 1635 twips, each row in the table
' is 330 twips tall.
   VerticalPlacement = 1635 + (330 * MatchedRoom)
' Create the text block to hold the reservation.
   Set Tt = New TextBox(CurrentView.Body)
' Fill the text block with the reservation owner's name
' and spaces to center the text properly.
   Tt.Text = " " + Txt + " "
' Set display properties for the text block to match the form.
   Tt.Font.Size = 8
   ' Use Approach LotusScript constants for border style.
   Tt.Border.Style = $ltsBorderStyleNone
   Tt.Border.Left = True
   Tt.Border.Right = True
   Tt.Border.Top = False
   Tt.Border.Bottom = False
   ' Use Approach constants for line width.
   Tt.Border.Width = $apr1point
   ' Use LotusScript constants for color.
   Call Tt.Border.Color.SetRGB(COLOR_ULTRAMARINE)
   Call Tt.Background.Color.SetRGB (COLOR_50_GRAY)
' Set the position of the text block to correspond to the
' correct room and time.
   Tt.Height = 325
   Tt.Top = VerticalPlacement   ' Current offset from top of
                                ' form
' Convert reservation time (passed in) to the horizontal
' location and length on the form.
   Tt.Left = (((start - 8) * 750) + 945)
   Tt.Width =  (750 * (finish - start))
```

```
' Add a prefix to the name of the text block so the
' ClearDisplay function can delete the reservation.
   Tt.Name = "Tt" + Str$(Tt.Top) + Str$(Tt.Left)
End Sub
```

How do I find records using the Find object?

There are two ways to automate finding records:

- Create a found set from records in a main or detail table associated with the specified .APR file. Specify the first find condition with the New method, and add more find conditions using the And or Or methods of the Find object.
- Create a result set by accessing data from any table through Connection, Query, and ResultSet objects. Specify find conditions using the SQL property of the Query object. The retrieved data is manipulated without appearing in the user interface.

  For more information about creating a result set, see Accessing data from a database using a batch process.

Before you choose which way to find data, consider how you want to use the found records:

- If you create a found set, you can see the record data immediately in fields in the application views.
- If you create a result set, you must do one of the following with the data:
  - Modify the data through LotusScript.
  - Display data from the result set in text blocks or other display elements.
  - Create a new .APR file based on the result set.

These techniques for working with a result set are described in How do I modify records using a result set.

Finding records using a Find object involves the following operations:

- Determining the find condition or conditions. To use input from the user for the find, display a form as a dialog box or use the LotusScript InputBox function.
- Creating a Find object, specifying the field to search and the value to match.
- Adding other conditions to the Find object, using the And or Or methods.

Example - button script to find records that match user input

The following example prompts the user for a last name and a state name, and it creates and executes a find using the input. The script is part of the sample application stored in C:\<*PATH*>\Samples\dw08_s2.APR. The script is attached to the Click event for a button on a form.

```
Sub Click(Source As Button, X As Long, Y As Long, _
   Flags As Long)
' Click event for the btnLast_St object
' Apply the Find and Sort objects to the DocWindow using the
' FindSort method.
' Prompt the user to enter an employee's last name and state.
' Start search after the user enters the information.
   ' Create a DocWindow object.
   Dim MyDocWin As DocWindow
   ' Retrieve the active DocWindow.
   Set MyDocWin = CurrentApplication.ActiveDocWindow
   ' Create a Table object.
   Dim  MyTable As String
   ' Retrieve the name of the first table for the document.
   MyTable = CurrentDocument.Tables(0).TableName
' Prompt the user to enter find conditions for Last name and
' State.
   Dim MyLast  As String   ' User input for Last name
   Dim MyState As String   ' User input for State
   ' Prompt user to enter employee's last name.
   MyLast = Inputbox$("Enter employee's last name", , ,300,300)
   ' Prompt user to enter postal code for the state.
   MyState = Inputbox$("Enter state abbreviation " & _
      "(For example, CA for California)", , ,300,300)
```

```
' Check that the user enters information or does not press
' Cancel.
If MyLast <> "" And MyState <> "" Then
' Find the records that match the user's input and
' sort them in ascending order by last name and first name.
   ' Create a new instance of Find object to search by last
   ' name entered by the user.
   Dim MyFind As New Find (MyTable & ".Last", MyLast)
   ' Also find by state.
   Call MyFind.AND (MyTable & ".ST", MyState)
   ' Create new instance of Sort object, sorted by last name.
   ' The constant LtsSortAscending indicates the sort
   ' direction.
   Dim MySort As New Sort (MyTable & ".Last", LTSSORTASCENDING)
   ' Also sort by first name in ascending order.
   Call MySort.ADD (MyTable & ".First", LTSSORTASCENDING)
   ' Start Find/Sort.
   ' Add error handling here to check for finds that return no
   ' records.
   ' Run the find.
   Call MyDocWin.FindSort (MyFind,MySort)
' Show the find results in a worksheet view.
   Set CurrentWindow.ActiveView = CurrentDocument.Worksheet~ 1
Else
  Exit Sub   ' Exit if user pressed Cancel or did not enter
             ' values.
End Sub   ' Click event for the btnLast_St object
```

How do I insert and use OLE controls?

Note This feature is not supported under OS/2.

If you have an OLE controls (OCX) embedded in an Approach form, you can access the object's methods and properties through LotusScript as you would any other object. The following example opens an OCX Web browser, called Sax Webster Control, in the Internet World Wide Web Sites SmartMaster application, and passes it a URL string. The Webster OCX opens the Web page associated with that URL string, and the user is able to navigate through the Web site. If the Webster browser is not loaded, an error message alerts the user and informs the user how to install the OCX.

To use an OCX in Approach, you must first install the control. For more information, search on "Custom Controls" in the Approach Help Index.

Example - displaying a web site using the Webster browser

```
Sub Click(Source As Textbox, X As Long, Y As Long, _
   Flags As Long)
' Click event for txtWebsterBrowser of the Found Set Report
' view to run the Webster browser
' * RUN-TIME DEPENDENCIES
' * Files: This script is part of the Internet World
' * Wide Web Sites SmartMaster application. It requires the SAX
' * Webster Control.
   Dim Rval As Integer   ' Return value
' If there is no browser installed, print a message
' to the user (segment below).
   On Error 11026 GoTo NoWebster
   ' Set the global strURL to the current listing.
   StrURL = "http://" & source.URL.text
   ' If the First Viewed date is blank, add today's date.
   If (Source.fldFirstDate.Text = "") Then
      ' Enter today's date in the FirstDate field box.
      Source.fldFirstDate.ReadOnly = False
      Source.fldFirstDate.Text = Str(Today())
      Source.fldFirstDate.ReadOnly = True   ' Set it back.
   End If
   ' Add today's date to the Last Viewed date.
   Source.fldLastDate.ReadOnly = False
   Source.fldLastDate.Text = Str(Today())
   Source.fldLastDate.ReadOnly = True
   ' Switch to the Webster Browser view.
   Set CurrentApplication.ActiveView = _
      CurrentDocument.Webster~ Browser
   ' Load the URL into the Webster OCX.
   Rval = CurrentView.Body.oleWebster.LoadPage(strURL, 0)
   ' Leave this sub.
   GoTo EndLoadPage
NoWebster:
   ' Warn the user that the Webster connection isn't working.
   Print Err
   Messagebox "You don't have the Webster browser " & _
      "installed. Go to Main Menu-Setup to install the " & _
      "Webster browser."
```

```
   Resume EndLoadPage
EndLoadPage:
End Sub
```

How do I modify records using a display element?

There are two ways to modify records in a database using LotusScript:

- using a display element
  Change the text property of a field box, drop-down box, radio button, or other display object. This method handles each field in each record individually. It is especially suited to entering specific user input into a database.
- using a result set
  Create a result set and modify data in fields without exposing the data to the user. This method is especially useful if you are modifying a large number of records or updating records with information that isn't unique to each record or doesn't require user input.

  For more information, see How do I modify records using a result set.

The technique demonstrated here assumes that the data you want to modify appears in a view.

Example - button script to modify field contents in a view

This script loops through each record in the current found set to make the modifications. The script is in the sample application stored in C:\<*PATH*>\Samples\dw08_s4.APR.

```
Sub Click(Source As Button, X As Long, Y As Long, _
   Flags As Long)
' Click event for the btnEnterComment button object
' Prompt the user for input.
' Append today's date to the user's comment.
' Append the comment to each record in the found set.
' * RUN-TIME DEPENDENCIES
' * Files: This script requires a field named Note in the main
' * table associated with the .APR file.
   ' Create a DocWindow object.
   Dim MyDocWin As DocWindow
   ' Retrieve the active DocWindow.
   Set MyDocWin = CurrentApplication.ActiveDocWindow
   Dim UserInput  As String     ' User input
   Dim NoteEntry As String      ' The input with today's date
   Dim PreviousEntries As String   ' Existing contents of Note
   Dim WholeNote As String         ' All the data from Note
   Dim I As Integer                ' Index of found set
   ' Get input from user.
   UserInput = Inputbox$("Enter your comments", , ,300,300)
   ' Check that the user entered a comment.
   If UserInput <> "" Then
      ' Append today's date to the user input.
      NoteEntry = Date$ & ": " & UserInput
      ' Loop through each record in the found set and
      ' update the Note field.
      MyDocWin.FirstRecord      ' Go to first record.
      ' Loop through the number of records in the found set.
      For I = 1 To MyDocWin.NumRecordsFound
         ' Store the existing contents of the Note field.
         PreviousEntries = Source.Note.Text
         ' Append the new entry to the existing ones.
         WholeNote = PreviousEntries & " " & NoteEntry & "."
         ' Insert the new Note in the field.
```

```
         Source.Note.Text = WholeNote
         ' Go to the next record in the found set.
         MyDocWin.NextRecord
      Next
   End If   ' If the user entered a comment.
End Sub   ' Click event for btnEnterComment
```

How do I modify records using a result set?

There are two ways to modify records in a database using LotusScript:

- using a result set
  Create a result set and modify data in fields without exposing the data to the user. This method is especially useful if you are modifying a large number of records or updating records with information that isn't unique to each record or doesn't require user input.
- using a display element
  Change the text property of a field box, drop-down box, radio button, or other display object. This method handles each field in each record individually. It is especially suited to entering specific user input into a database.

  For more information, see How do I modify records using a display element .

The technique demonstrated here requires you to create a result set using Connection and Query objects as described in Accessing data from a database using a batch process. After you create the result set, loop through the records in the result set and make the changes.

Example - button script to modify field contents from a result set

The following example illustrates this process. The text of this script is stored in C:\<*PATH*>\Samples\dw08_s3.lss.

```
Sub Click(Source As Button, X As Long, Y As Long, _
   Flags As Long)
' Click event for any button object
' * RUN-TIME DEPENDENCIES
' * Files: This script requires an ODBC database named Sample
' * in the same directory as the .APR file. The database
' * contains a table named USERID.CUSTOMER. The table contains
' * the fields State and SaleRep.
   Dim fName As String
   Dim Con As New Connection
   Dim Qry As New Query
   Dim Rs As New ResultSet
   Dim ChkSetV As Integer
   Dim MyVal As Variant
   Dim I As integer   ' Index to table rows
   Dim TextToMatch As String
' Determine the find condition.
' Note: Here you can use some value in the current .APR to
' evaluate changes in the other file.
' For example, use "If MyVal = Val(Source.field1.text)  Then"
TextToMatch = "CA"
' Open a connection to the table.
   If (Con.ConnectTo ("ODBC Data Sources","userid", _
      "password", "!Sample")<>False) Then
   ' Use this connection for the query.
      Set Qry.Connection = con
      ' Specify the table for the query.
      Qry.TableName = "USERID.CUSTOMER"
   ' Use this query to create the result set.
      Set Rs.Query = Qry
   ' Create the result set.
      ' If the query was successful, then...
      If ((Rs.Execute)<>False) Then
         ' Find the number of columns (fields) in the table.
```

```
         N = Rs.NumColumns
         ' Print the number of columns to the IDE Output panel.
         Print "Number of columns = ", N
         ' Print the names of each field in the table.
         For I= 1 To N
            fName = Rs.Fieldname (I)
            Print fName   ' Output appears in the IDE.
         Next
      Else   ' If the result set was not successful, warn the
             ' user.
         Messagebox "The query did not succeed.", _
            MB_OK + MB_ICONINFORMATION + MB_APPLMODAL, _
           "Unsuccessful Query"
      End If   ' If the result set was successful.
   Else   ' If the connection was not successful, warn the
          ' user.
      Messagebox "Connection failed", MB_OK + _
         MB_ICONINFORMATION + MB_APPLMODAL, "Connection"
   End If   ' If the connection was successful.
' Read the value in a field and check if it matches the value.
' Continue while the variable i is less than or equal to the
' number of rows in the table.
   For I = 1 To Rs.NumRows
      Print I
      ' Get the value of the field State.
      MyVal = Rs.GetValue("State")
      Print MyVal
      ' If the value in the field matches the find condition,
      ' then set the value of another field.
      If MyVal = TextToMatch Then
         ' Set the field "SalesRep" to the value SF.
         Rs.SetValue "SalesRep", "SF"
      End If      ' The field value matches.
      ' Update the current row and move to the next one.
      ' UpdateRow is required to commit the changes for each
      ' record.
      Rs.UpdateRow
      Rs.NextRow
   Next      ' Until there are no more rows in the table
' Disconnect here to avoid trouble reconnecting later.
   Con.Disconnect
End Sub
```

How do I read information from the Registry?

The Registry stores information about your system, the software that you have installed, your user preferences, the names of installed components, and your network services such as your Internet connections. Reading values in the Registry and using them in your scripts can be quite useful.

The following scripts illustrate how to access the Registry to get the file path for an installed SmartSuite product. To modify these scripts to get different infromation in the Registry, do the following:

1. Run the utility named REGEDIT.EXE.
2. Browse the Registry for the infromation that you want to extract. Note the precise specification of the Registry entry, such as HKEY_LOCAL_MACHINE\Software\Lotus\SmartSuite\97.0 in the left pane, and the name of the value to be extracted, such as "Path", in the right pane.
3. Use this specification to modify the values of the following in the sub GetLotusProductPath:
   - KeyName$ variable
   - ValueName$ variable
   - First argument to the function RegOpenKeyExA

Example - global script

Copy these declarations to your global (Declarations) or !Declarations script in your document.

```
' (Declarations) or !Declarations
' Declarations for functions to read the Registry.
Declare Public Function RegOpenKeyExA Lib "advapi32" _
   Alias "RegOpenKeyExA" _
   (Byval HKEY As Long,Byval lpszSubKey As String, _
   Byval dwreserved As Integer,Byval samDesired As Long, _
   keyresult As Long) As Long
Declare Public Function RegQueryValueExA Lib "advapi32" _
   Alias "RegQueryValueExA" _
   (Byval HKEY As Long,Byval lpszValueName As String, _
   Byval dwreserved As Integer, lpdwtype As Long, _
   Byval lpData As String, readbytes As Long) As Long
Declare Public Function RegCloseKey Lib "advapi32" _
   Alias "RegCloseKey" (Byval HKEY As Long) As Long
```

Example - document script

Copy this sub to your document.

```
Sub CallingSub
   ' Declare a variable for the return value of the function GetProductPath.
    Dim autoPath As String
   ' Get the value of GetLotusProductPath.
    autoPath = GetLotusProductPath
    MsgBox "The path for the product = " & autopath
End Sub
```

Example - document function

Copy this sub to your document.

```
Function GetLotusProductPath As String
   ' This function reads the 95 Registry and returns the
   ' file path for the specified SmartSuite product.
   ' Some variables needed by the functions calling the Registry.
   Dim happkey As Long
```

```
   Dim HKEY_LOCAL_MACHINE As Long
   Dim KEY_QUERY_VALUE As Long
   Dim KEY_READ As Long
   Dim KEY_ENUMERATE_SUBKEYS As Long
   Dim KEY_NOTIFY As Long
   Dim KeyName As String
   Dim ValueName As String
   Dim ValueType As Long
   Dim readbytes As Long
   Dim lstat As Long
   ' Some assignments for the functions.
   HKEY_LOCAL_MACHINE= &H80000002
   KEY_QUERY_VALUE=1
   KEY_ENUMERATE_SUBKEYS=8
   KEY_NOTIFY=16
   KEY_READ=KEY_QUERY_VALUE Or KEY_ENUMERATE_SUBKEYS Or KEY_NOTIFY

   ' A variable to store the base path to the Lotus Suite.
   Dim LotusProductPath As String
   LotusProductPath$=String(255,Chr$(32))

   ' The following key identifies the SmartSuite 97 product to look
   ' in the Registry.
   ' Uncomment the appropriate key.
   KeyName$="Software\Lotus\Approach\97.0"
'  KeyName$="Software\Lotus\123\97.0"
'  KeyName$="Software\Lotus\Freelance\97.0"
'  KeyName$="Software\Lotus\WordPro\97.0"

   ' The following key identifies the name of the Registry value to look up.
   ' This is the same for all our products.
   ValueName$="Path"

   ' Reads the Registry and returns the path to the SmartSuite product
   ' as LotusProductPath$.
   lstat=RegOpenKeyExA(HKEY_LOCAL_MACHINE,KeyName$,0,KEY_READ,happkey)
   ReadBytes=255
   lstat=RegQueryValueExA(happkey,ValueName$,0, _
      valueType, LotusProductPath$,ReadBytes)
   regclosekey(happkey)

   ' Trim some trailing spaces.
   GetLotusProductPath=Left$(LotusProductPath$,ReadBytes-1)
End Function
```

How do I switch between views in a document?

One of the most common ways to control the flow of an application is to create a view for each task that a user may perform in the application. Provide a menu view to guide users through the tasks. For each task, attach a script (or a macro) to a button in the menu view that switches to a view that you have set up for the task.

You can easily automate switching views with a macro. If switching views is only one out of a series of actions being automated, a script may be more suitable.

The following example comes from the Approach Meeting Room Scheduler SmartMaster application. To open this SmartMaster, choose File - New Database and select the Meeting Room Scheduler. The script appears in the Click event for the btnToday object on the Start view. When the user clicks this button, the script is executed and the following occurs:

- The application switches to the Schedule Display view in order to show the rooms and times that are reserved for a specific date.
- The current date (from the system) is formatted and appears in a text block at the top of the view.
- Schedule information in the view from previous uses is cleared.
- Today's schedule information appears in the view.

The following scripts accomplish this sequence of operations.

Example - button script calling a global function

This script appears in the Click event of the btnToday object on the Start view.

```
Sub Click(Source As Button, X As Long, Y As Long, _
   Flags As Long)
   ' Display the schedule for the current system date.
   Call DisplaySchedule(Format$(Now, "m/d/yy"))
End Sub
```

Example - global function for displaying a view

The function DisplaySchedule is a global function in the same .APR file. The tilde (~) precedes a space in the name of a view.

```
Function DisplaySchedule(DateToDisplay As String)
   ' DateToDisplay   Schedule date to display information for.
   ' Change to the Schedule Display view.
   Set CurrentWindow.ActiveView = _
      CurrentDocument.Schedule~ Display
   ' Display the passed-in date in fbxDateDisplay field box.
   CurrentView.Body.fbxDateDisplay.Text = DateToDisplay
   ' Clear schedule information from the view.
   Call ClearDisplay()      ' Global sub
   ' Fill the schedule information for the date.
   Call ReadBlock(DateToDisplay)   ' Global function
End Function
```