SIEBEL VB LANGUAGE REFERENCE

VERSION 7.5, REV. C

12-FRKIMH

JULY 2003

Contents

Introduction

Chapter 1. Quick Reference: Statements and Functions

Chapter 2. Language Overview

Chapter 3. Siebel VB Language Reference

Appendix A. Siebel VB Compared to Other Basic Products

Introduction

Siebel VB is an enhanced configuration environment which includes:

- A fully functional procedural programming language
- A bidirectional application interface to provide bidirectional access to Siebel Business Objects
- An editing environment to create and maintain custom Siebel VB routines
- A debugger to assist in detecting errors in Siebel VB routines
- A compiler to compile the custom Siebel VB routines
- A run-time engine (similar to a Basic interpreter) to process the custom Siebel VB routines

You can use Siebel VB to create scripts that automate a variety of daily tasks.

Developers looking for scripting functionality on their UNIX-hosted Siebel Object Managers should read *Siebel eScript Language Reference*.

Supported Uses of Siebel VB

This document describes the supported functionality of the Siebel VB language and provides examples of how a Siebel developer uses Siebel VB. Siebel eBusiness Applications provide a high performance client/server application specifically designed to meet the most rigorous sales and marketing information requirements of large multi-national corporations. Caution should be exercised when extending the Siebel Sales Enterprise application, which should be done only by trained technical professionals.

NOTE: Improper application configuration can adversely effect the reliability and performance characteristics of your configured Siebel application. Thorough testing is strongly recommended before production rollout of your configured application.

In summary Siebel VB supports:

- Siebel VB language to behave as documented
- Siebel Tools for creating, modifying, and deleting of Siebel VB scripts as documented in *Siebel Object Interfaces Reference*.

Siebel VB does not support:

- Functionality developed through custom programming
- Automatic upgrades of custom routines with the Siebel Application Upgrader
- Development of separate, standalone applications with Siebel VB
- Accessing server management functions through Siebel VB; such functions should be accessed only through the UI or the command line

NOTE: Siebel VB is not supported in a UNIX environment.

How This Guide Is Organized

This guide is divided as follows.

Chapter 1, “Quick Reference: Statements and Functions” offers a summary of the commands and functions, divided into functional areas. It also contains brief descriptions of each command and function.

Chapter 2, “Language Overview” describes the essential rules and components of Siebel VB.

Chapter 3, “Siebel VB Language Reference” contains a full listing of every command and function, including examples, in Siebel VB. The Appendixes provide information about trappable error codes and derived trigonometric functions, and comparisons between Siebel VB and other versions of Basic.

Typographic Conventions

This guide uses the following typographic conventions.

Table 1. Typographic Conventions

To Represent	Help Syntax Is
Statements and functions	Initial letter uppercase: Abs Len(*variable*)
Arguments to statements or functions	Lowercase, italicized letters; an internal capital may be used to indicate multiple English words: *variable*, *rate*, *prompt*, *sringVar*
Optional arguments and/or characters	Arguments and/or characters in brackets: [, *caption*], [*type*], [$]
Required choice for an argument from a list of choices	A list inside braces, with OR operator (|) separating choices: {Goto label | Resume Next | Goto 0}

Revision History

Siebel VB Language Reference, Version 7.5, Rev. C

Version 7.5, Rev. C

Table 2. Changes Made in Rev. C

Topic	Revision
"Arrays" on page 163	Modified explanation of element counts.
"Declare Statement" on page 155	Modified Usage information.
"Input Function" on page 261	Modified Usage information.
"Open Statement" on page 344	Modified Usage information.
ShowStatus Method	Removed. This method has been deprecated.

Version 7.5, Rev. B

Table 3. Changes Made in Rev. B

Topic	Revision
"Call Statement" on page 112	Updated usage information.
"Language Overview" on page 47	Added information on date comparisons.
"Now Function" on page 331	Improved the Returns information.
"SQLRetrieve Function" on page 454	Added limitation information on Destination().
"Strings" on page 56	Added information on CRLF issues.
"Type Statement" on page 487	Updated usage information.

Additional Changes:

- Modified examples to remove errors.

Version 7.5, Rev. A

Table 4. Changes Made in Rev. A

Topic	Revision
"Object Handling" on page 66	Added text on good programming practice.
"SQLOpen Function" on page 448	Repaired syntax error in the example.
"SQLRequest Function" on page 451	Repaired syntax error in the example.

Additional Changes:

- Modified examples to remove deprecated commands.

Quick Reference: Statements and Functions 1

This quick reference lists the Siebel VB statements and functions by functional group.

Arrays

The following functions and statements are used for manipulating arrays.

Function or Statement	Purpose	For More Information
Erase	Reinitializes the contents of an array	"Erase Statement" on page 175
LBound	Returns the lower bound of an array's dimension	"LBound Function" on page 287
ReDim	Declares dynamic arrays and reallocates memory	"ReDim Statement" on page 377
UBound	Returns the upper bound of an array's dimension	"UBound Function" on page 490

Compiler Directives

The following statements are compiler directives.

Function or Statement	Purpose	For More Information
Rem	Treats the remainder of the line as a comment	"Rem Statement" on page 380
'	Treats the remainder of the line as a comment	"Rem Statement" on page 380
_	Treats the next line as a continuation of the current line	

Control Flow

The following statements control the logic flow.

Statement	Purpose	For More Information
Call	Transfers control to a subprogram	“Call Statement” on page 112
Do...Loop	Controls repetitive actions	“Do...Loop Statement” on page 169
Exit	Causes the current procedure or loop structure to return	“Exit Statement” on page 187
For...Next	Loops a fixed number of times	“For...Next Statement” on page 199
Goto	Sends control to a line label	“GoTo Statement” on page 253
If...Then...Else	Branches on a conditional value	“If...Then...Else Statement” on page 259
Let	Assigns a value to a variable	“Let (Assignment Statement)” on page 293
Lset	Left-aligns one string or a user-defined variable within another	“Lset Statement” on page 307
On...Goto	Branches to one of several labels depending upon value	“On...GoTo Statement” on page 340
Rset	Right-aligns one string or a user-defined variable within another	“Reset Statement” on page 385
Select Case	Executes one of a series of statement blocks	“Select Case Statement” on page 403
Set	Sets an object variable to a value	“Set Statement” on page 411
Stop	Stops program execution	“Stop Statement” on page 461

Statement	Purpose	For More Information
While...Wend	Controls repetitive actions	"While...Wend Statement" on page 507
With	Executes a series of statements on a specified variable or object	"With Statement" on page 510

Dates and Times

The following functions and statements are for use with date and time information.

Function or Statement	Purpose	For More Information
Date Function	Returns the current date	"Date Function" on page 146
Date Statement	Sets the computer's date	"Date Statement" on page 147
DateSerial	Returns the date value for year, month, and day specified	"DateSerial Function" on page 149
DateValue	Returns the date value for the string specified	"DateValue Function" on page 151
Day	Returns the day of month component of a date-time value	"Day Function" on page 153
Hour	Returns the hour of day component of a date-time value	"Hour Function" on page 257
IsDate	Determines whether a value is a legal date	"IsDate Function" on page 276
Minute	Returns the minute component of a date-time value	"Minute Function" on page 316
Month	Returns the month component of a date-time value	"Month Function" on page 320
Now	Returns the current date and time	"Now Function" on page 331
Second	Returns the second component of a date-time value	"Second Function" on page 397
Time Function	Returns the current time	"Time Function" on page 473
Time Statement	Sets the current time	"Time Statement" on page 475

Function or Statement	Purpose	For More Information
Timer	Returns the number of seconds since midnight	"Timer Function" on page 477
TimeSerial	Returns the time value for the hour, minute, and second specified	"TimeSerial Function" on page 479
TimeValue	Returns the time value for the string specified	"TimeValue Function" on page 481
Weekday	Returns the day of the week for the specified date-time value	"Weekday Function" on page 505
Year	Returns the year component of a date-time value	"Year Function" on page 515

Declarations

The following statements are for data declarations.

Statement	Purpose	For More Information
Const	Declares a symbolic constant	"Const Statement" on page 131
Declare	Forward declares a procedure in the same module or in a dynamic-link library	"Declare Statement" on page 155
Def*type*	Declares the default data type for variables	"Deftype Statement" on page 158
Dim	Declares variables	"Dim Statement" on page 161
Function...End Function	Defines a function	"Function...End Function Statement" on page 213
Global	Declares a global variable	"Global Statement" on page 249
Option Base	Declares the default lower bound for array dimensions	"Option Base Statement" on page 347
Option Compares	Declares the default case sensitivity for string comparisons	"Option Compare Statement" on page 350
Option Explicit	Forces variables to be declared explicitly	"Option Explicit Statement" on page 352
ReDim	Declares dynamic arrays and reallocates memory	"ReDim Statement" on page 377
Static	Defines a static variable or subprogram	"Static Statement" on page 460

Statement	Purpose	For More Information
Sub...End Sub	Defines a subprogram	“Sub...End Sub Statement” on page 467
Type	Declares a user-defined data type	“Type Statement” on page 487

Environment Control

The following functions and statements relate to the computer's environment.

Function or Statement	Purpose	For More Information
AppActivate	Activates another application	"SendKeys Statement" on page 405
Command	Returns the command line specified when the MAIN sub was run	
Date Statement	Sets the current date	"Date Statement" on page 147
Environ	Returns a string from the operating system's environment	"Environ Function" on page 171
Randomize	Initializes the random-number generator	"Randomize Statement" on page 373
SendKeys	Sends keystrokes to another application	"SendKeys Statement" on page 405
Shell	Runs an executable program	"Shell Function" on page 432

Errors

The following functions and statements relate to error handling.

Function or Statement	Purpose	For More Information
Erl	Returns the line number where a run-time error occurred	"Erl Function" on page 177
Err Function	Returns a run-time error code	"Err Function" on page 179
Err Statement	Sets the run-time error code	"Err Statement" on page 180
Error Function	Returns a string representing an error	"Error Function" on page 182
Error Statement	Generates an error condition	"Error Statement" on page 184
On Error	Controls run-time error handling	"On Error Statement" on page 341
Resume	Ends an error-handling routine	"Resume Statement" on page 387
Trappable Errors	Errors that can be trapped by Siebel VB code	"Trappable Errors" on page 523

Files: Disk and Folder Control

The following functions and statements relate to folders.

Function or Statement	Purpose	For More Information
ChDir	Changes the default folder for a drive	"ChDir Statement" on page 118
ChDrive	Changes the default drive	"ChDrive Statement" on page 120
CurDir	Returns the current folder for a drive	"CurDir Function" on page 141
Dir	Returns a filename that matches a pattern	"Dir Function" on page 167
MkDir	Creates a folder on a disk	"MkDir Statement" on page 318
RmDir	Removes a folder from a disk	"RmDir Statement" on page 390

Files: File Control

The following functions and statements are for file control.

Function or Statement	Purpose	For More Information
FileAttr	Returns information about an open file	"FileAttr Function" on page 190
FileCopy	Copies a file	"FileCopy Statement" on page 192
FileDateTime	Returns the modification date and time of a specified file	"FileDateTime Function" on page 194
FileLen	Returns the length of a specified file in bytes	"FileLen Function" on page 195
GetAttr	Returns attributes of a specified file, folder, or volume label	"GetAttr Function" on page 222
Kill	Deletes files from a disk	"Kill Statement" on page 284
Name	Renames a disk file	"Name Statement" on page 323
SetAttr	Sets attribute information for a file	"SetAttr Statement" on page 413

Files: File Input/Output

The following functions and statements are for file input and output.

Function or Statement	Purpose	For More Information
Close	Closes a file	“Close Statement” on page 129
Eof	Checks for end of file	“Eof Function” on page 173
FreeFile	Returns the next unused file number	“FreeFile Function” on page 212
Get	Reads bytes from a file	“Get Statement” on page 218
Input Function	Returns a string of characters from a file	“Input Function” on page 261
Input Statement	Reads data from a file or from the keyboard	“Input Statement” on page 262
Line Input	Reads a line from a sequential file	“Line Input Statement” on page 297
Loc	Returns the current position in an open file	“Loc Function” on page 299
Lock	Controls access to some or all of an open file by other processes	“Lock Statement” on page 300
Lof	Returns the length of an open file	“Lof Function” on page 302
Open	Opens a disk file for I/O	“Open Statement” on page 344
Print	Prints data to a file or to the screen	“Print Statement” on page 364
Put	Writes data to an open file	“Put Statement” on page 367

Function or Statement	Purpose	For More Information
Reset	Closes all open disk files	"Reset Statement" on page 385
Seek Function	Returns the current position for a file	"Seek Function" on page 399
Seek Statement	Sets the current position for a file	"Seek Statement" on page 401
Spc	Outputs a given number of spaces	"Spc Function" on page 435
Tab	Moves the print position to the given column	"Tab Function" on page 469
Unlock	Controls access to some or all of an open file by other processes	"Unlock Statement" on page 494
Width	Sets the output-line width for an open file	"Width Statement" on page 509
Write	Writes data to a sequential file	"Write Statement" on page 512

Math Functions: Financial Functions

The following functions are for calculating financial information.

Function	Purpose	For More Information
FV	Returns future value of a cash flow stream	“FV Function” on page 216
IPmt	Returns interest payment for a given period	“IPmt Function” on page 271
IRR	Returns internal rate of return for a cash flow stream	“IRR Function” on page 273
NPV	Returns net present value of a cash flow stream	“NPV Function” on page 333
Pmt	Returns a constant payment per period for an annuity	“Pmt Function” on page 356
PPmt	Returns principal payment for a given period	“PPmt Function” on page 361
PV	Returns present value of a future stream of cash flows	“PV Function” on page 369
Rate	Returns interest rate per period	“Rate Function” on page 375

Math Functions: Numeric Functions

The following functions are for mathematics.

Function	Purpose	For More Information
Abs	Returns the absolute value of a number	“Abs Function” on page 76
Exp	Returns the value of *e* raised to a power	“Exp Function” on page 188
Fix	Returns the integer part of a number	“Fix Function” on page 197
Int	Returns the integer part of a number	“Int Function” on page 268
IsNumeric	Determines whether a value is a legal number	“IsNumeric Function” on page 283
Log	Returns the natural logarithm of a value	“Log Function” on page 303
Rnd	Returns a random number	“Rnd Function” on page 392
Sgn	Returns a value indicating the sign of a number	“Sgn Function” on page 430
Sqr	Returns the square root of a number	“Sqr Function” on page 459

Math Functions: Trigonometric Functions

The following functions are for trigonometric calculations.

Function	Purpose	For More Information
Atn	Returns the arc tangent of a number	"Atn Function" on page 90
Cos	Returns the cosine of an angle	"Cos Function" on page 133
Sin	Returns the sine of an angle	"Sin Function" on page 433
Tan	Returns the tangent of an angle	"Tan Function" on page 471
Derived Functions	How to compute other trigonometric functions	"Derived Trigonometric Functions" on page 527

Objects

The following commands and statements are for object control.

Command or Statement	Purpose	For More Information
Clipboard	Accesses the Windows Clipboard	"Clipboard" on page 126
CreateObject	Creates a COM automation object	"CreateObject Function" on page 134
GetObject	Retrieves a COM object from a file or gets the active COM object for a COM class	"GetObject Function" on page 234
Is	Determines whether two object variables refer to the same object	"Is Operator" on page 275
Me	Gets the current object	"Me" on page 311
New	Allocates and initializes a new COM object	"New Operator" on page 325
Nothing	Sets an object variable to not refer to an object	"Nothing Function" on page 329
Object	Declares a COM automation object	"Object Class" on page 336
Typeof	Checks the class of an object	"Typeof Function" on page 489
With	Executes statements on an object or a user-defined type	"With Statement" on page 510

ODBC

The following functions are for data access.

Function	Purpose	For More Information
SQLClose	Closes a data source connection	"SQLClose Function" on page 437
SQLError	Returns a detailed error message (ODBC functions)	"SQLError Function" on page 439
SQLExecQuery	Executes a SQL statement	"SQLExecQuery Function" on page 442
SQLGetSchema	Obtains information about data sources, databases, terminology, users, owners, tables, and columns	"SQLGetSchema Function" on page 445
SQLOpen	Establishes a connection to a data source for use by other functions	"SQLOpen Function" on page 448
SQLRequest	Makes a connection to a data source, executes a SQL statement, returns the results	"SQLRequest Function" on page 451
SQLRetrieve	Returns the results of a select statement that was executed by SQLExecQuery into a user-provided array	"SQLRetrieve Function" on page 454
SQLRetrieveToFile	Returns the results of a select statement that was executed by SQLExecQuery into a user-specified file	"SQLRetrieveToFile Function" on page 457

Strings: String Functions

The following functions and statements are for string manipulation.

Function or Statement	Purpose	For More Information
GetField	Returns a substring from a delimited source string	“GetFieldValue Method” on page 227
Hex	Returns the hexadecimal representation of a number, as a string	“Hex Function” on page 256
InStr	Returns the position of one string within another	“InStr Function” on page 265
LCase	Converts a string to lower case	“LCase Function” on page 289
Left	Returns the left portion of a string	“Left Function” on page 290
Len	Returns the length of a string or size of a variable	“Len Function” on page 292
Like Operator	Compares a string against a pattern	“Like Operator” on page 295
LTrim	Removes leading spaces from a string	“LTrim Function” on page 309
Mid Function	Returns a portion of a string	“Mid Function” on page 312
Mid Statement	Replaces a portion of a string with another string	“Mid Statement” on page 314
Oct	Returns the octal representation of a number, as a string	“Oct Function” on page 338
Right	Returns the right portion of a string	“Right Function” on page 388
RTrim	Removes trailing spaces from a string	“RTrim Function” on page 396

Function or Statement	Purpose	For More Information
SetField	Replaces a substring within a delimited target string	"SetFieldValue Method" on page 414
Space	Returns a string of spaces	"Space Function" on page 434
Str	Returns the string representation of a number	"Str Function" on page 462
StrComp	Compares two strings	"StrComp Function" on page 463
String	Returns a string consisting of a repeated character	"String Function" on page 465
Trim	Removes leading and trailing spaces from a string	"Trim Function" on page 486
UCase	Converts a string to upper case	"UCase Function" on page 492

Strings: String Conversions

The following functions are for string conversion.

Function	Purpose	For More Information
Asc	Returns an integer corresponding to a character code	"Asc Function" on page 88
CCur	Converts a value to currency	"CCur Function" on page 115
CDbl	Converts a value to double-precision floating point	"CDbl Function" on page 117
Chr	Converts a character code to a string	"Chr Function" on page 121
CInt	Converts a value to an integer by rounding	"CInt Function" on page 123
CLng	Converts a value to a long by rounding	"CLng Function" on page 128
CSng	Converts a value to single-precision floating point	"CSng Function" on page 137
CStr	Converts a value to a string	"CStr Function" on page 139
CVar	Converts a number or string to a variant	"CVar Function" on page 143
CVDate	Converts a value to a variant date	"CVDate Function" on page 144
Format	Converts a value to a string using a picture format	"Format Function" on page 202
Val	Converts a string to a number	"Val Function" on page 495

Variants

The following functions are for variant data.

Function	Purpose	For More Information
IsEmpty	Determines whether a variant has been initialized	"IsEmpty Function" on page 277
IsNull	Determines whether a variant contains a NULL value	"IsNull Function" on page 281
Null	Returns a null variant	"Null Function" on page 334
VarType	Returns the type of data stored in a variant	"VarType Function" on page 497

Language Overview 2

If you have never programmed in Visual Basic before, you may find the following hints helpful.

Declare your variables. As a general rule, use the Option Explicit statement, because it forces you to declare your variables (using the Dim statement) before you use them. Declaring your variables makes it easier for others to understand your code, and for you to debug the code. You can declare a variable without giving it a data type. If you do not specify a data type, Siebel VB assumes the type Variant, which requires 16 bytes—twice as much memory as the next smallest data type. If you can avoid using Variant variables, you reduce the amount of memory required by your code, which may make execution faster. In Siebel VB, you place Option commands in the (general) (declarations) window.

Use standardized naming conventions. Another way to improve the readability of your code is to follow a set of standardized naming conventions. It does not matter what conventions you follow as long as everyone in the programming group follows the same conventions. One very common convention is to prefix each variable with a letter denoting its type, as shown here.

Data Type	Symbol	Example
String	`s`	`sName`
Integer	`i`	`iReturn`
Long integer	`l`	`lBigCount`
Single-precision number	`si`	`siAllowance`
Double-precision number	`d`	`dBudget`
Object	`o`	`oBusComp`
Currency	`c`	`cAmtOwed`

You can also use suffix characters on your variable names, as described in "Type Characters" on page 57.

The Me object reference. The special object reference *Me* is a VB shorthand for "the current object." Use it in place of references to active Siebel business objects. For example, in a business component event handler, you should use Me in place of *ActiveBusComp,* as shown in the following example.

```
Function BusComp_PreSetFieldValue(FieldName As String, FieldValue
As String) As Integer
   If Val(Me.GetFieldValue("Rep %")) >75 Then
   ....
   End If
   BusComp_PreSetFieldValue = ContinueOperation
End Function
```

Trap errors. Especially in a LAN environment, where you cannot be sure that a record has not been changed or deleted by another user, you should create routines that keep the program from failing when it meets an unexpected condition. For more information about error-handling routines, read "Error Handling" on page 68.

Make effective use of the Select Case construct. Use the Select Case construct to choose among any number of alternatives your require, based on the value of a single variable. This is preferable to a series of nested If statements, because it simplifies code maintenance, and also improves performance, because the variable must be evaluated only once. For a full description of the Select Case construct, read "Select Case Statement" on page 403.

Use the With shortcut. Use the With statement to apply several methods to a single object. It makes the code easier to read, reduces typing, and improves performance. Instead of a series of statements such as

```
Set oBusComp = objBusObject.GetBusComp("Opportunity")
oBusComp.ClearToQuery
oBusComp.SetSearchSpec . . .
oBusComp.ExecuteQuery ForwardBackward
oBusComp.FirstRecord
oBusComp.NewRecord NewAfter
oBusComp.SetFieldValue "QuoteNumber", sQuoteId
oBusComp.SetFieldValue "Account", sAccount
. . .
sSolutionId(cSolution) = oBusComp.GetFieldValue( "Id" )
. . .
```

use the following:

```
Set oBusComp = objBusObject.GetBusComp("Opportunity")
With oBusComp
   .ClearToQuery
   .SetSearchSpec . . .
   .ExecuteQuery ForwardBackward
   .FirstRecord
   .NewRecord NewAfter
   .SetFieldValue "QuoteNumber", sQuoteId
   .SetFieldValue "Account", sAccount
   . . .
   sSolutionId(cSolution) =.GetFieldValue( "Id" )
   . . .
End With
```

Use extreme care when working with date variables. When working with date variables, be careful with the date format. GetFieldValue returns the date in the format dd/mm/yyyy (followed by the time). The CVDate() function expects the regional setting. As a result, applying the CVDate() function to the return value may cause an error. The GetFormattedFieldValue method uses the regional settings of the user's operating system. The regional settings specify the year with two digits in most cases, thereby creating the possibility of Y2K noncompliance. For these reasons, you should use the following approach for performing date arithmetic:

- Retrieve the value of date fields with the GetFieldValue method. For more information, read *Siebel Object Interfaces Reference*.
- Convert it into a date variable using the DateSerial function.
- Perform the required date arithmetic.

Here is an example:

```
Dim strDate as String, varDate as Variant
strDate = oBC.GetFieldValue("Date Field")
varDate = DateSerial(Mid(strDate,7,4), Left(strDate,2), _
   Mid(strDate,4,2))
[any date arithmetic]
```

When comparing date values, use the DateSerial function on the date values first. This makes sure that the values are in the same format so that the comparison is valid. Date values from different sources may be in different formats. DateSerial provides a uniform format for all dates. For example, you are checking to see if an employee's hire date is before a specific benefits changeover date. You should use the DateSerial function on both the hire date and the benefits changeover date, and then you can make a valid comparison between the two date values because they are in the same format.

Conventions

The following describes the programming conventions used by Siebel VB.

- "Arguments" on page 51
- "Named Arguments" on page 52
- "Comments" on page 53

Arguments

Arguments to subprograms and functions you write are listed after the subroutine or function and might or might not be enclosed in parentheses. Whether you use parentheses depends on whether you want to pass the argument to the subroutine or function by value or by reference.

If you pass an argument by value, the variable used for that argument retains its value when the subroutine or function returns to the caller. If you pass an argument by reference, the variable's value may be changed for the calling procedure. For example, suppose you set the value of a variable, x, to 5 and pass x as an argument to a subroutine, named mysub. If you pass x by value to mysub, then x remains 5 after mysub returns. If you pass x by reference to mysub, however, then x can be changed by mysub and may have a different value.

NOTE: Siebel VB functions support a maximum of 32 arguments. If you need to use more than 32 arguments, use the Type function to define a custom data type and pass arguments of this new type.

To pass an argument by value, use one of the following syntax options:

```
Call mysub((x))

mysub(x)

y = myfunction((x))

Call myfunction((x))
```

To pass an argument by reference, use one of the following options:

```
Call mysub(x)

mysub x

y = myfunction(x)

Call myfunction(x)
```

Externally declared subprograms and functions (such as DLL functions) can be declared to take byVal arguments in their declaration. In that case, those arguments are always passed byVal.

NOTE: Array variables cannot be passed to externally declared subprograms and functions.

Named Arguments

When you call a subroutine or function that takes arguments, you usually supply values for those arguments by listing them in the order shown in the syntax for the statement or function. For example, suppose you define a function this way:

```
myfunction(id, action, value)
```

From the preceding syntax, you know that the function called myfunction requires three arguments: *id*, *action*, and *value*. When you call this function, you supply those arguments in the order shown. If the function contains just a few arguments, you can remember their order with ease. However, if a function has several arguments, and you want to be sure the values you supply are assigned to the correct arguments, use named arguments.

Named arguments are arguments that are identified by name rather than by position in the syntax. To use a named argument, use the following syntax:

```
namedarg:= value
```

Using this syntax for myfunction, you get:

```
myfunction id:=1, action:="get", value:=0
```

The advantage of named arguments is that you do not need to remember the original order as they were listed in the syntax, so the following function call is also correct:

```
myfunction action:="get", value:=0, id:=1
```

With named arguments, order is not important.

The area in which named arguments have a significant advantage is in calls to functions or subprograms that have a mix of required and optional arguments. Ordinarily, you need to use commas as placeholders in the syntax for the optional arguments that you do not use. However, with named arguments you can specify just the arguments you want to use and their values without regard to their order in the syntax. For example, if myfunction is defined as:

```
myfunction(id, action, value, Optional counter)
```

you can use named arguments as follows:

```
myfunction id:="1", action:="get", value:="0"
```

or

```
myfunction value:="0", counter:="10", action:="get", id:="1"
```

NOTE: Although you can shift the order of named arguments, you cannot omit required arguments. Siebel VB functions and statements accept named arguments. The argument names are listed in the syntax for each statement and function.

Comments

Comments are preceded by an apostrophe and can appear on their own line in a procedure or directly after a statement or function on the same line:

```
' This comment is on its own line

Dim i as Integer ' This comment is on the code line
```

Comments are also indicated by the Rem statement.

Data Types

Basic is a strongly typed language. Variables can be declared implicitly on first reference by using a type character. If no type character is present, the default type of Variant is assumed. Alternatively, the type of a variable can be declared explicitly with the Dim statement. In either case, the variable can contain data only of the declared type. Variables of a user-defined type *must* be explicitly declared. Siebel VB supports standard Basic numeric, string, record, and array data. Siebel VB also supports Dialog Box Records and Objects (which are defined by the application).

- "Arrays" on page 54
- "Numbers" on page 55
- "Records" on page 56
- "Strings" on page 56
- "Type Characters" on page 57

Arrays

Arrays are created by specifying one or more subscripts at declaration or when the array is redimensioned by the ReDim statement (read "ReDim Statement" on page 377). Subscripts specify the beginning and ending index for each dimension. If only an ending index is specified, the beginning index depends on the Option Base setting. Array elements are referenced by enclosing the proper number of index values in parentheses after the array name. For example, *arrayName(i,j,k)* indicates an array with three dimensions. For more information, read "Dim Statement" on page 161.

For examples of the use of arrays, read "IsEmpty Function" on page 277, "IsNull Function" on page 281, "NPV Function" on page 333, "Null Function" on page 334, "Option Base Statement" on page 347, and "VarType Function" on page 497.

Numbers

Table 5 shows the numeric types.

Table 5. Numeric Types

Type	Description	From	To
Integer	2-byte integer	-32,768	32,767
Long	4-byte integer	-2,147,483,648	2,147,483,647
Single	4-byte floating-point number	-3.402823e+38 0.0, 1.401298e-45	-1.401298e-45, 3.402823466e+38
Double	8-byte floating-point number	-1.797693134862315d+308, 0.0, 2.2250738585072014d-308	-4.94065645841247d-308, 1.797693134862315d+308
Currency	8-byte number with a fixed decimal point	-922,337,203,685,477.5808	922,337,203,685,477.5807

Numeric values are always signed.

Siebel VB has no true Boolean variables. Basic considers 0 to be FALSE and any other numeric value to be TRUE. Only numeric values can be used as Booleans. Comparison operator expressions always return 0 for FALSE and -1 for TRUE.

Integer constants can be expressed in decimal, octal, or hexadecimal notation. Decimal constants are expressed by simply using the decimal representation. To represent an octal value, precede the constant with `&O` or `&o` (for example, `&o177`). To represent a hexadecimal value, precede the constant with `&H` or `&h` (for example, `&H8001`).

Records

A record, or record variable, is a data structure containing one or more elements, each of which has a value. Before declaring a record variable, a Type must be defined. When the Type is defined, the variable can be declared to be of that type. The variable name should not have a type character suffix. Record elements are referenced using dot notation, for example,

```
record.element
```

where *record* is the previously defined record name and *element* is a member of that record. Records can contain elements that are themselves records.

Strings

Siebel VB strings can be either fixed or dynamic. Fixed strings have a length specified when they are defined, and the length cannot be changed. Fixed strings cannot be of 0 length. Dynamic strings have no specified length. Any string can vary in length from 0 to 32,767 characters. There are no restrictions on the characters that can be included in a string. For example, the character whose ANSI value is 0 can be embedded in strings.

NOTE: You can use characters only from the current character set. Within a character set, any character can be embedded either by cutting and pasting or by using the Chr function. For more information, read "Chr Function" on page 121.

When exchanging data with other applications, be aware of terminating characters. Siebel VB terminates its output text with a carriage return and a line feed (CRLF), and expects the same characters on input (unless specifically noted for some input functions). Some applications generate and expect only carriage returns.

Type Characters

Siebel VB permits the use of special characters as the suffix to the name of a function, variable, or constant. The character defines the data type of the variable or function, and operates as a *de facto* declaration. The type characters are shown in Table 6.

Table 6. Data Type Suffix Characters

Data Type	Suffix
Dynamic String	$
Integer	%
Long Integer	&
Single-precision floating-point	!
Double-precision floating-point	#
Currency (exact fixed point)	@

Data Type Conversions

Numeric conversions. Siebel VB converts data between any two numeric types. When converting from a larger type to a smaller type (for example, Long to Integer), a run-time numeric overflow may occur. This indicates that the number of the larger type is too large for the target data type. Imprecision is not a run-time error (for example, when converting from Double to Single, or from either float type to either integer type).

String conversions. Siebel VB also converts between fixed strings and dynamic strings. When converting from a fixed a string to a dynamic string, it creates a dynamic string that has the same length and contents as the fixed string. When converting from a dynamic string to a fixed string, some adjustment may be necessary. If the dynamic string is shorter than the fixed string, the resulting fixed string is extended with spaces. If the dynamic string is longer than the fixed string, the resulting fixed string is a truncated version of the dynamic string. No run-time errors are caused by string conversions.

Variant conversions. Siebel VB converts between any data type and variants. Basic converts variant strings to numbers when required. A type mismatch error occurs if the variant string does not contain a valid representation of the required number.

No other implicit conversions are supported. In particular, Siebel VB does not convert automatically between numeric and string data. Use the Val function to convert string to numeric data, and the Str function to convert numeric to string data.

Dynamic Arrays

Dynamic arrays differ from fixed arrays because a subscript range for the array elements is not specified when the array is dimensioned. Instead, the subscript range is set using the ReDim statement. With dynamic arrays, the number of array elements can be set based on other conditions in your procedure. For example, you may want to use an array to store a set of values entered by the user, but you may not know in advance how many values the user will enter. In this case, you dimension the array without specifying a subscript range and then execute a ReDim statement each time the user enters a new value. Or you may want to prompt for the number of values to be entered and execute one ReDim statement to set the size of the array before prompting for the values.

If you use ReDim to change the size of an array and want to preserve the contents of the array at the same time, be sure to include the Preserve argument to the ReDim statement:

```
Redim Preserve ArrayName(n)
```

The following procedure uses a dynamic array, *varray*, to hold cash flow values entered by the user:

```
Sub main
   Dim aprate as Single
   Dim varray() as Double
   Dim cflowper as Integer
   Dim msgtext as String
   Dim x as Integer
   Dim netpv as Double
   cflowper=2
   ReDim varray(cflowper)
   For x= 1 to cflowper
   varray(x)=500
   Next x
   aprate=10
   If aprate>1 then
      aprate=aprate/100
   End If
   netpv=NPV(aprate,varray())
   msgtext="The net present value is: "
   msgtext=msgtext & Format(netpv, "Currency")
   TheApplication.raiseErrorText msgtext
End Sub
```

If you declare a dynamic array (with a Dim statement) before using it, the maximum number of dimensions it can have is 8. To create dynamic arrays with more dimensions (up to 60), do not declare the array at all and use only the ReDim statement inside your procedure.

Variant Data Type

The variant data type can be used to define variables that contain any type of data. A tag is stored with the variant data to identify the type of data that it currently contains. You can examine the tag by using the VarType function.

A variant can contain a value of any of the types listed in Table 7.

Table 7. Variant Value Types

Type/Name	Size of Data	Range
0 (Empty)	0	N/A
1 Null	0	N/A
2 Integer	2 bytes (short)	-32768 to 32767
3 Long	4 bytes (long)	-2.147E9 to 2.147E9
4 Single	4 bytes (float)	-3.402E38 to -1.401E-45 (negative) 1.401E-45 to 3.402E38 (positive)
5 Double	8 bytes (double)	-1.797E308 to -4.94E-324 (negative) 4.94E-324 to 1.797E308 (positive)
6 Currency	8 bytes (fixed)	-9.223E14 to 9.223E14
7 Date	8 bytes (double)	Jan 1st, 100 to Dec 31st, 9999
8 String	up to 2GB	Length is limited by the amount of random access memory, up to 2 GB
9 Object	N/A	N/A

A newly defined Variant defaults to being of Empty type to signify that it contains no initialized data. An Empty Variant is converted to zero when used in a numeric expression, or to an empty string when used in a string expression. To test whether a variant is uninitialized (empty), use the IsEmpty function.

Null variants have no associated data and serve only to represent invalid or ambiguous results. You can test whether a variant contains a null value with the IsNull function. Null is not the same as Empty, which indicates that a variant has not yet been initialized.

Expressions

An expression is a collection of two or more terms that perform a mathematical or logical operation. The terms are usually either variables or functions that are combined with an operator to evaluate to a string or numeric result. You use expressions to perform calculations, manipulate variables, or concatenate strings.

Expressions are evaluated according to precedence order. Use parentheses to override the default precedence order.

The precedence order (from high to low) for the operators is:

- "Numeric Operators" on page 63
- "String Operators" on page 64
- "Comparison Operators (Numeric and String)" on page 64
- "Logical Operators" on page 65

Numeric Operators

Operator	Comments
^	Exponentiation.
-, +	Unary minus and plus.
*, /	Numeric multiplication or division. For division, the result is a Double.
\	Integer division. The operands can be Integer or Long.
Mod	Modulus or Remainder. The operands can be Integer or Long.
-, +	Numeric addition and subtraction. The + operator can also be used for string concatenation.

String Operators

Operator	Comments
&	String concatenation
+	String concatenation

Comparison Operators (Numeric and String)

Operator	Comments
>	Greater than
<	Less than
=	Equal to
< =	Less than or equal to
> =	Greater than or equal to
< >	Not equal to

For numbers, the operands are widened to the least common type (Integer is preferable to Long; Long is preferable to Single; Single is preferable to Double). For Strings, the comparison is case-sensitive, and is based on the collating sequence used by the language specified by the user using the Windows Control Panel. The result is 0 for FALSE and -1 for TRUE.

Logical Operators

Operator	Comments
NOT	Unary Not—operand can be Integer or Long. The operation is performed bitwise (one's complement).
AND	And—operands can be Integer or Long. The operation is performed bitwise.
OR	Inclusive Or—operands can be Integer or Long. The operation is performed bitwise.
XOR	Exclusive Or—operands can be Integer or Long. The operation is performed bitwise.
EQV	Equivalence—operands can be Integer or Long. The operation is performed bitwise. (A EQV B) is the same as (NOT (A XOR B)).
IMP	Implication—operands can be Integer or Long. The operation is performed bitwise. (A IMP B) is the same as ((NOT A) OR B).

Object Handling

Objects are reusable blocks of code that can be instantiated or referenced to perform specific tasks. They may be the end products of a software application, such as a spreadsheet, graph, or document. Each software application has its own set of properties and methods that change the characteristics of an object.

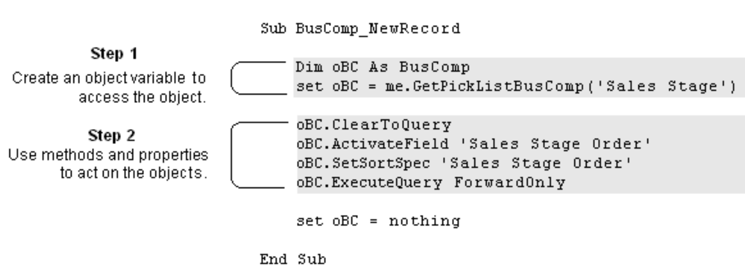
Properties affect how an object behaves. For example, width is a property of a range of cells in a spreadsheet, colors are a property of graphs, and margins are a property of word processing documents.

Methods cause the application to perform an action on an object. Examples are Calculate for a spreadsheet, Snap to Grid for a graph, and Auto-Save for a document.

In Siebel VB, you can access Siebel objects and change the properties and methods of that object. This means that you can access an object that is part of the Siebel application by running a VB program external to the Siebel program.

However, before you can use a non-Siebel object in a Siebel VB procedure, you must instantiate the object by assigning it to an object variable. Then attach an object name (with or without properties and methods) to the variable to manipulate the object.

Alternatively, when accessing Siebel objects within Siebel VB you can declare an object as one of the supported Siebel object types. The syntax for doing this is shown in the following example code.



In this example, oBC is not declared “as Object”, but rather is declared “as BusComp”. Here you are instantiating one of the Siebel object types, the business component (BusComp) object type. You could declare it as an object, but if you want to use the methods associated with the object type, as shown in step 2, you must declare it as the appropriate object type.

Finally, the preceding example ends by setting oBC to “nothing”. In keeping with good programming practices, always set an object to “nothing” when it has been instantiated.

NOTE: You can use a similar procedure to access other types of COM-compliant objects. Use the original software application that created them to change properties and methods of the objects. You can see an example in “CreateObject Function” on page 134.

Creating an Object Variable to Access the Object

The Dim statement creates an object variable called *oBC* and assigns a picklist business component to it. The Set statement assigns the business component to the variable oBC using a Get method. If you are instantiating an application, use either GetObject or CreateObject. Use GetObject if the application is already open on the Windows desktop. Use CreateObject if the application is not open.

Using Methods and Properties to Act on Objects

To access an object, property or method, use this syntax:

```
appvariable.object.property
appvariable.object.method
```

For example, me.GetPickListBusComp(“Sales Stage”) is a value returned by the GetPickListBusComp method of the BusComp object for the Siebel application, which is assigned to the object variable oBC.

Error Handling

Siebel VB contains three error handling statements and functions for trapping errors in your program: Err, Error, and On Error. Siebel VB returns a code for many of the possible run-time errors you might encounter. For a list of codes, read "Trappable Errors" on page 523

In addition to the errors trapped by Siebel VB, you may want to create your own set of codes for trapping errors specific to your program. For example, create your own set of codes if your program establishes rules for file input and the user does not follow the rules. You can trigger an error and respond appropriately using the same statements and functions you would use for error codes returned by Siebel VB.

Regardless of the error trapped, you can use two methods to handle errors. You can put error-handling code directly before a line of code where an error might occur (such as after a File Open statement), or you can label a separate section of the procedure just for error handling, and force a jump to that label if any error occurs. The On Error statement handles both options.

For more information, read "Trapping Errors Returned by Siebel VB" on page 69 and "Trapping User-Defined, Non-Siebel VB Errors" on page 71.

Trapping Errors Returned by Siebel VB

This code example shows the two ways to trap errors. Option 1 places error-handling code directly before the line of code that could cause an error. Option 2 contains a labeled section of code that handles any error.

"Option 1: Trap Errors Within Body of Code" on page 70

"Option 2: Trap Errors Using an Error Handler" on page 70

Option 1: Trap Errors Within Body of Code

The On Error statement identifies the line of code to go to in case of an error. In this case, the Resume Next parameter means that execution continues with the next line of code after the error. In this example, the line of code to handle errors is the If statement. It uses the Err statement to determine which error code is returned.

Option 1

Place error-handling code within the body of a procedure.

Option 2

Place error-handling code within the body of a procedure and Goto it using a label.

```
Sub Main
    Dim UserDrive As String, UserDir As String
    Dim msgtext As string
in1:
    UserDrive = "C:"
    On Error Resume Next
    Err = 0
    ChDrive UserDrive
    If Err = 68 Then
      TheApplication.RaiseErrorText "Invalid drive.
      Try again."
      Goto in1
    End If

    On Error Goto Errortrap1
in2:
    UserDir = "test"
    ChDir UserDrive & "\" & UserDir
    Exit Sub

Errortrap1:
   Select Case Err
        Case 75
                msgtext = "Path is invalid."
        Case 76
                msgtext = "Path not found."
        Case Else
                msgtext = "Error " & Err & ":" & Error$
    End Select
    TheApplication.RaiseErrorText msgtext & " Try again."
    Resume in2
End Sub
```

Option 2: Trap Errors Using an Error Handler

The On Error statement used here specifies a label to jump to in case of errors. The code segment is part of the main procedure and uses the Err statement to determine which error code is returned. To make sure your code does not accidentally fall through to the error handler, precede it with an Exit statement.

Trapping User-Defined, Non-Siebel VB Errors

These code examples show the two ways to set and trap user-defined errors. Both options use the Error statement to set the user-defined error to the value 30000. To trap the error, option 1 places error-handling code directly before the line of code that could cause an error. Option 2 contains a labeled section of code that handles any user-defined errors.

Option 1

Place error-handling code within the body of a procedure.

```
Sub Main
     Dim custname As String
     On Error Resume Next
inl:
     Err = 0
     custname = ""
     If custname = "" Then
     Error 30000
     Select Case Err
          Case 30000
               TheApplication.RaiseErrorText "You must enter a
               customer name"
               Goto inl
          Case Else
               TheApplication.RaiseErrorText "Undetermined error.
               Try again"
     End Select
     End If
     TheApplication.RaiseErrorText " The Name is " &
     custname & "."
End Sub
```

Option 2

Place error-handling code within the body of a procedure and Goto it using a label.

```
Sub Main
     Dim custname As String
     On Error Goto Errortrap1
in1:
     Err = 0
     custname = ""
     If custname = "" Then
          Error 30000
     End If
     TheApplication.RaiseErrorText " The Name is " &
     custname & "."
     Exit Sub
Errortrap1:
     Select Case Err
          Case 30000
               TheApplication.RaiseErrorText "You must enter a
               customer name"
          Case Else
               TheApplication.RaiseErrorText "Undetermined
               error. Try again"
          End Select
     Resume in1
End Sub
```

Trapping Errors Generated by Siebel VB Methods

Many Siebel VB methods return error codes, but they must be handled differently from those returned by the standard VB functions and statements. Siebel VB methods use numeric error codes in the range from 4000 to 4999. For errors generated by Siebel VB methods, use a construct of this form to see the text of the error message:

```
DisplayError:
   If ErrCode <> 0 Then
      ErrText = GetLastErrText
      TheApplication.RaiseErrorText ErrText
      Exit Sub
   End If
```

For more information, read *Siebel Object Interfaces Reference*.

Note that `DisplayError:` is a label and is the target of a Goto statement elsewhere in the program.

NOTE: The GetLastErrText method is available only through interfaces external to Siebel Tools. Therefore, you can use it in Microsoft VB, but not in Siebel VB.

Siebel VB and Unicode

Siebel VB supports Unicode with the following exceptions. Functions that provide File Input/Output or which access external DLLs are code page-dependent and not Unicode-compliant.

Siebel VB Language Reference 3

This language reference lists the Siebel VB statements and functions in alphabetical order, and indicates:

- Syntax
- Return value
- Usage
- An example
- A list of related commands

This reference also includes information about Siebel VB methods and events. Siebel VB methods are used to access and affect components of the Siebel software architecture such as applets and business components. Siebel VB methods must be prefaced by the name of the architecture component to be addressed; for example:

```
BusComp.GetFieldValue(fieldName)
```

Standard VB commands do not address specific components of the Siebel software architecture. In this guide, standard VB functions and statements and Siebel VB methods are always identified as such in the description. For details about Siebel VB events and methods, read *Siebel Object Interfaces Reference*.

Abs Function

This standard VB function returns the absolute value of a number.

Syntax Abs(*number*)

Argument	Description
number	Any valid numeric expression

Returns The absolute value of *number*.

Usage The data type of the return value matches the type of the *number*. If *number* is a variant string (vartype 8), the return value is converted to vartype 5 (double). If the absolute value evaluates to vartype 0 (Empty), the return value is vartype 3 (long).

Example This example finds the difference between two variables, `oldacct` and `newacct`.

```
Sub Button_Click
Dim oldacct, newacct, count
   oldacct = 1234566
   newacct = 33345
   count = Abs(oldacct - newacct)
End Sub
```

See Also "Exp Function" on page 188
"Fix Function" on page 197
"Int Function" on page 268
"Log Function" on page 303
"Rnd Function" on page 392
"Sgn Function" on page 430
"Sqr Function" on page 459

ActivateField Method

ActivateField allows queries to retrieve data for the argument-specified field. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

ActivateMultipleFields Method

ActivateMultipleFields allows users to activate the fields specified in the property set input argument. For details, read *Siebel Object Interfaces Reference*.

ActiveBusObject Method

ActiveBusObject returns the business object for a Siebel business component for the active applet. This method is used with the application object. For details, read *Siebel Object Interfaces Reference*.

ActiveViewName Method

ActiveViewName returns the name of the active view. It is used with applet and application objects. For details, read *Siebel Object Interfaces Reference*.

AddChild Method

Use the AddChild method to add subsidiary property sets to a property set to form tree-structured data structures. For details, read *Siebel Object Interfaces Reference*.

Application_Close Event

The Close Siebel VB event handler is called before exiting the application and after the PreClose event. This allows Basic scripts to perform last minute cleanup (such as cleaning up a connection to a COM server). It is called when the application is notified by Windows that it should close, but not if the process is terminated directly. For details, read *Siebel Object Interfaces Reference*.

Application_InvokeMethod Event

The InvokeMethod Siebel VB event handler is called after a specialized method is invoked. For details, read *Siebel Object Interfaces Reference*.

Application_Navigate Event

The Navigate event is called after the client has navigated to a view. For details, read *Siebel Object Interfaces Reference*.

Application_PreInvokeMethod Event

The PreInvokeMethod Siebel VB event handler is called before a specialized method is invoked by a user defined applet menu or by calling InvokeMethod on the application. For details, read *Siebel Object Interfaces Reference*.

Application_PreNavigate Event

The PreNavigate() event is called before the client has navigated from one view to the next. For details, read *Siebel Object Interfaces Reference*.

Application_Start Event

The Application_Start event handler is called when the user has successfully logged into the application. For details, read *Siebel Object Interfaces Reference*.

Asc Function

This standard VB function returns an integer corresponding to the ANSI code of the first character in the specified string.

Syntax Asc(*string*)

Argument	Description
string	A string expression of one or more characters

Returns An integer corresponding to the ANSI code of the first character in the argument.

Usage To change an ANSI code to string characters, use Chr.

Example This example asks the user for a letter and returns its ANSI value.

```
Sub Button_Click
   Dim userchar As String
   Dim ascVal as Integer
   userchar = "Z"
   ascVal = Asc(userchar)
End Sub
```

See Also "Chr Function" on page 121

Associate Method

The Associate method creates a new many-to-many relationship for the parent object through an association business component. It is used with business components. For details, read *Siebel Object Interfaces Reference*.

Atn Function

This standard VB function returns the angle (in radians) for the arctangent of the specified number.

Syntax Atn(*number*)

Argument	Description
number	Any valid numeric expression

Returns The arctangent of *number*.

Usage The Atn function assumes *number* is the ratio of two sides of a right triangle: the side opposite the angle to find and the side adjacent to the angle. The function returns a single-precision value for a ratio expressed as an integer, a currency, or a single-precision numeric expression. The return value is a double-precision value for a long, variant, or double-precision numeric expression.

To convert radians to degrees, multiply by (180/PI). The value of PI is approximately 3.14159.

Example This example finds the roof angle necessary for a house with an attic ceiling of 8 feet (at the roof peak) and a 16-foot span from the outside wall to the center of the house. The Atn function returns the angle in radians; it is multiplied by 180/PI to convert it to degrees.

```
Sub Button_Click
   Dim height As Single, span As Single, angle As Single
   Dim PI As Single
   PI = 3.14159
   height = 8
   span = 16
   angle = Atn(height/span) * (180/PI)
End Sub
```

See Also

"Cos Function" on page 133
"Sin Function" on page 433
"Tan Function" on page 471
"Derived Trigonometric Functions" on page 527

BusComp Method

The BusComp Siebel VB method returns a Siebel Business Component that is associated with an object. It is used with applet objects, and business objects. For details, read *Siebel Object Interfaces Reference*.

BusComp_Associate Event

The Associate Siebel VB event handler is called after a record is added to a business component to create an association. For details, read *Siebel Object Interfaces Reference*.

BusComp_ChangeRecord Event

The ChangeRecord Siebel VB event handler is called after a record becomes the current row in a Siebel business component. For details, read *Siebel Object Interfaces Reference*.

BusComp_CopyRecord Event

The CopyRecord Siebel VB event handler is called after a row has been copied in a Siebel business component and that row has been made active. For details, read *Siebel Object Interfaces Reference*.

BusComp_DeleteRecord Event

The DeleteRecord Siebel VB event handler is called after a row in a Siebel business component is deleted. The current context is a different row (the Fields of the just-deleted row are no longer available). For details, read *Siebel Object Interfaces Reference*.

BusComp_InvokeMethod Event

The InvokeMethod Siebel VB event handler is called when a specialized method is called on a Siebel business component. For details, read *Siebel Object Interfaces Reference*.

BusComp_NewRecord Event

The NewRecord Siebel VB event handler is called after a new row has been created in a Siebel business component and that row has been made active. The event may be used to set up default values for Fields. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreAssociate Event

The PreAssociate Siebel VB event handler is called before a record is added to a Siebel business component to create an association. The semantics are the same as BusComp_PreNewRecord. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreCopyRecord Event

The PreCopyRecord Siebel VB event handler is called before a new row is copied in a Siebel business component. The event may be used to perform precopy validation. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreDeleteRecord Event

The PreDeleteRecord Siebel VB event handler is called before a row is deleted in a Siebel business component. The event may be used to prevent the deletion or to perform any actions in which you need access to the record that is to be deleted. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreGetFieldValue Event

The PreGetFieldValue Siebel VB event handler is called when the value of a business component field is accessed. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreInvokeMethod Event

The PreInvokeMethod Siebel VB event handler is called before a specialized method is invoked on a Siebel business component. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreNewRecord Event

The PreNewRecord Siebel VB event handler is called before a new row is created in a Siebel business component. The event may be used to perform preinsert validation. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreQuery Event

The PreQuery Siebel VB event handler is called before query execution. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreSetFieldValue Event

The PreSetFieldValue Siebel VB event handler is called before a value is pushed down into a Siebel business component from the user interface or through a call to SetFieldValue. For details, read *Siebel Object Interfaces Reference*.

BusComp_PreWriteRecord Event

The PreWriteRecord Siebel VB event handler is called before a row is written out to the database. The event may perform any final validation necessary before the actual save occurs. For details, read *Siebel Object Interfaces Reference*.

BusComp_Query Event

The Query Siebel VB event handler is called just after the query is done and the rows have been retrieved, but before the rows are actually displayed. For details, read *Siebel Object Interfaces Reference*.

BusComp_SetFieldValue Event

The SetFieldValue Siebel VB event handler is called when a value is pushed down into a Siebel business component from the user interface or through a call to SetFieldValue. For details, read *Siebel Object Interfaces Reference*.

BusComp_WriteRecord Event

The WriteRecord Siebel VB event handler is called after a row is written out to the database. For details, *Siebel Object Interfaces Reference*.

BusObject Method

BusObject returns the business object for the specified object. It is used with applet, service, control, and web applet objects. For details, read *Siebel Object Interfaces Reference*.

Call Statement

This standard VB function transfers control to a subprogram or function.

Syntax A Call *subprogram_name* [(*argument_list*)]

Syntax B *subprogram_name argument_list*

subprogram_name is the name of the subprogram or function to which control is to be passed.

Argument	Description
argument_list	The arguments, if any, to be passed to the subroutine or function

Returns If a function, its output; if a subprogram, not applicable.

Usage Use the Call statement to call a subprogram or function written in Basic or to call C procedures in a DLL. These C procedures must be described in a Declare statement or be implicit in the application. Make sure the DLL is present on every Siebel Server.

If a procedure accepts named arguments, you can use the names to specify the argument and its value. Order is not important. For example, if a procedure is defined as follows:

```
Sub mysub(aa, bb, optional cc, optional dd)
```

The following calls to this procedure are equivalent to each other:

```
call mysub(1, 2, , 4)
mysub aa :=  1, bb :=  2, dd := 4
call mysub(aa :=  1, dd:= 4, bb :=  2)
mysub 1, 2, dd:= 4
```

The syntax for named arguments is as follows:

```
argname := argvalue
```

where *argname* is the name for the argument as supplied in the Sub or Function statement and *argvalue* is the value to assign to the argument when you call it.

The advantage to using named arguments is that you do not have to remember the order specified in the procedure's original definition, and if the procedure takes optional arguments, you do not need to include commas (,) for arguments that you leave out.

The procedures that can use named arguments include:

- Functions defined with the Function statement.
- Subprograms defined with the Sub statement.
- Procedures declared with Declare statement.
- Many built-in functions and statements.
- Some externally registered DLL functions and methods.

Arguments are passed *by reference* to procedures written in Basic. If you pass a variable to a procedure that modifies its corresponding formal parameter, and you do not want to have your variable modified (that is, if you need to retain the "before" value), enclose the variable in parentheses in the Call statement. This tells Siebel VB to pass a copy of the variable. (This is called passing *by value*.) Note, however, that generally passing by value is less efficient, and should not be done unless necessary.

When a variable is passed to a procedure that expects its argument by reference, the variable must match the exact type of the formal parameter of the function. (This restriction does not apply to expressions or variants.)

When calling an external DLL procedure, arguments can be passed by value rather than by reference. This is specified in the Declare statement, the Call statement itself, or both, using the ByVal keyword. If ByVal is specified in the declaration, then the ByVal keyword is optional in the call. If present, it must precede the value. If ByVal was not specified in the declaration, it is illegal in the call unless the data type was unspecified in the declaration.

Example This example calls a subprogram named `CreateFile` to open a file, write the numbers 1 to 10 in it, and leave it open. The calling procedure then checks the file's mode. If the mode is 1 (open for Input) or 2 (open for Output), the procedure closes the file.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   Rem Put the numbers 1-10 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 1 to 10
      Write #1, x
   Next x
End Sub

Sub Button1_Click
   Dim filemode as Integer
   Dim attrib as Integer
   Call CreateFile
   attrib = 1
   filemode = FileAttr(1,attrib)
   If filemode = 1 or filemode = 2 then
      Close #1
   End If
   Kill "c:\temp001"
End Sub
```

See Also "Declare Statement" on page 155

CCur Function

This standard VB function converts an expression to the data type currency.

Syntax CCur(*expression*)

Argument	Description
expression	Any expression that evaluates to a number

Returns The value of *expression* as a number of type currency.

Usage CCur accepts any type of expression. Numbers that do not fit in the currency data type result in an Overflow error. Strings that cannot be converted result in a Type Mismatch error. Variants containing null result in an Illegal Use of Null error.

Example This example converts a yearly payment on a loan to a currency value with four decimal places. A subsequent Format statement formats the value to two decimal places before displaying it in a message box.

```
Sub Button_Click
Dim aprate, totalpay, loanpv
   Dim loanfv, due, monthlypay
   Dim yearlypay, msgtext
   loanpv = 5000
   aprate = 6.9
   If aprate >1 then
      aprate = aprate/100
   End If
   aprate = aprate/12
   totalpay = 360
   loanfv = 0
Rem Assume payments are made at end of month
   due = 0
   monthlypay = Pmt(aprate,totalpay,-loanpv,loanfv,due)
   yearlypay = CCur(monthlypay * 12)
   msgtext = "The yearly payment is: " & _
      Format(yearlypay, "Currency")
End Sub
```

See Also

"CDbl Function" on page 117
"Chr Function" on page 121
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144

CDbl Function

This standard VB function converts an expression to the data type double.

Syntax CDbl(*expression*)

Argument	Description
expression	Any expression that evaluates to a number

Returns The value of *expression* as a double-precision number.

Usage CDbl accepts any type of expression. Strings that cannot be converted to a double-precision floating point result in a Type Mismatch error. Variants containing null result in an Illegal Use of Null error.

Example This example calculates the square root of 2 as a double-precision floating-point value and displays it in scientific notation.

```
Sub Button_Click
Dim value
   Dim msgtext
   value = CDbl(Sqr(2))
   msgtext = "The square root of 2 is: " & Value
End Sub
```

See Also
"CCur Function" on page 115
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144

ChDir Statement

This standard VB statement changes the default folder for the specified drive.

Syntax ChDir [*drive*][[\]*folder*\]*folder*

Argument	Description
drive	The name of the drive containing the desired default folder as a letter, or a string expression representing the drive name; a colon is not required
[\]*folder*\	If the folder is not within the current folder of the specified drive (or the default drive if none is specified), the path to the folder to become the default, or a string expression representing the path
folder	The name of the folder to become the default, or a string expression representing the folder name

Returns Not applicable

Usage If the drive argument is omitted, ChDir changes the default folder on the current drive. If the initial backslash in [\]*folder*\ is omitted, ChDir changes to a folder within the current folder. If it is included, the path is followed from the root folder.

The ChDir statement does not change the default drive. To change the default drive, use ChDrive.

Example This example changes the current folder to `C:\Windows`, if it is not already the default.

```
Sub Button_Click
   Dim newdir as String
   newdir = "c:\Windows"
   If CurDir  <>  newdir then
      ChDir newdir
   End If
End Sub
```

See Also

"ChDrive Statement" on page 120
"CurDir Function" on page 141
"Dir Function" on page 167
"MkDir Statement" on page 318
"RmDir Statement" on page 390

ChDrive Statement

This standard VB statement changes the default drive.

Syntax ChDrive *drive*

Argument	Description
drive	A string expression designating the new default drive

Returns Not applicable

Usage A colon is permitted but not required as part of the name of the drive; a single letter suffices. The drive to be made the default must exist and must be within the range specified by the LASTDRIVE statement in the `config.sys` file. If a null string (`""`) is supplied as the argument, the default drive remains the same. If the *drive* argument is a string, ChDrive uses the first letter only. If the argument is omitted, an error message is displayed. To change the current folder on a drive, use ChDir.

Example This example changes the default drive to A.

```
Sub Button_Click
   Dim newdrive as String
   newdrive = "A"
   If Left(CurDir,2)  <>  newdrive then
      ChDrive newdrive
   End If
End Sub
```

See Also "ChDir Statement" on page 118
"CurDir Function" on page 141
"Dir Function" on page 167
"MkDir Statement" on page 318
"RmDir Statement" on page 390

Chr Function

This standard VB function returns the one-character string corresponding to an ANSI code.

Syntax Chr[$](*charCode*)

Argument	Description
charCode	An integer between 0 and 255 representing the ANSI code for a character

Returns The character represented by *charcode*.

Usage The dollar sign ($) in the function name is optional. If it is included, the return type is string; otherwise the function returns a variant of vartype 8 (string).

Example This example displays the character equivalent for an ASCII code between 65 and 122 typed by the user.

```
Sub Button_Click
   Dim numb as Integer
   Dim msgtext as String
   Dim out as Integer
   out = 0
   Do Until out
      numb = 75
      If Chr$(numb)> = "A" AND Chr$(numb)< = "Z" _
         OR Chr$(numb)> = "a" AND Chr$(numb)< = "z" then
         msgtext = "The letter for the number " & numb  _
            &" is: " & Chr$(numb)
         out = 1
      ElseIf numb = 0 then
         Exit Sub
      Else
         msgtext = "Does not convert to a character; try again."
      End If
   Loop
End Sub
```

See Also

"Asc Function" on page 88
"CCur Function" on page 115
"CDbl Function" on page 117
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144
"Format Function" on page 202
"Val Function" on page 495

CInt Function

This standard VB function converts an expression to the data type integer by rounding.

Syntax CInt(*expression*)

Argument	Description
expression	Any expression that evaluates to a number

Returns The value of *expression* as an integer.

Usage After rounding, the resulting number must be within the range of -32767 to 32767, or an error occurs.

Strings that cannot be converted to an integer result in a Type Mismatch error. Variants containing null result in an Illegal Use of Null error.

Example This example calculates the average of ten golf scores.

```
Sub Button_Click
   Dim score As Integer
   Dim x, sum
   Dim msgtext
   Let sum = 0
   For x = 1 to 10
      score = 7-
      sum = sum + score
   Next x
      msgtext = "Your average is: " & _
         Format(CInt(sum/ (x - 1)), "General Number")
End Sub
```

See Also

“CCur Function” on page 115
“CDbl Function” on page 117
“CLng Function” on page 128
“CSng Function” on page 137
“CStr Function” on page 139
“CVar Function” on page 143
“CVDate Function” on page 144

ClearToQuery Method

The ClearToQuery method clears the current query and sort specifications on a Siebel business component. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Clipboard

The Clipboard methods are standard VB methods that allow you to access the Windows Clipboard as an object.

Syntax

Clipboard.Clear
Clipboard.GetText()
Clipboard.SetText *string*
Clipboard.GetFormat()

Argument	Description
string	A string or string expression containing the text to send to the Clipboard

Returns Not applicable

Usage The Windows Clipboard can be accessed directly by your program so you can transfer text to and from other applications that support the Clipboard. The supported Clipboard methods are as follows:

Method	Description
Clear	Clears the contents of the Clipboard
GetText	Returns a text string from the Clipboard
SetText	Puts a text string on the Clipboard
GetFormat	Returns TRUE (non-zero) if the format of the item on the Clipboard is text; otherwise, returns FALSE (0)

NOTE: Data on the Clipboard is lost when another set of data of the same format is placed on the Clipboard (either through code or through a cut or copy operation in an application).

Example This example places the text string “Hello, world.” on the Clipboard.

```
Sub Button_Click
   Dim mytext as String
   mytext = "Hello, world."
   Clipboard.Settext mytext
End Sub
```

CLng Function

This standard VB function converts an expression to the data type long by rounding.

Syntax CLng(*expression*)

Argument	Description
expression	Any expression that can evaluate to a number

Returns The value of *expression* as a number of type long.

Usage After rounding, the resulting number must be within the range of -2,147,483,648 to 2,147,483,647, or an error occurs.

Strings that cannot be converted to a long result in a Type Mismatch error. Variants containing null result in an Illegal Use of Null error.

Example This example divides the US national debt by the number of people in the country to find the amount of money each person would have to pay to wipe it out. This figure is converted to a long integer and formatted as currency.

```
Sub Button_Click
   Dim debt As Single
   Dim msgtext
   Const Populace = 250000000
   debt = 8000000000000
   msgtext = "The $/citizen is: " & _
      Format(CLng(Debt/ Populace), "Currency")
End Sub
```

See Also
"CCur Function" on page 115
"CDbl Function" on page 117
"CInt Function" on page 123
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144

Close Statement

This standard VB statement closes a file, concluding input/output to that file.

Syntax Close [[#]*filenumber* [, [#]*filenumber* ...]]

Argument	Description
filenumber	The file number used in the Open statement to open the file, identifying the file to close

Returns Not applicable

Usage *Filenumber* is the number assigned to the file in the Open statement and can be preceded by a pound sign (#). If this argument is omitted, every open file is closed. When a Close statement is executed, the association of a file with *filenumber* is ended, and the file can be reopened with the same or a different file number.

When the Close statement is used, the final output buffer is written to the operating system buffer for that file. Close frees the buffer space associated with the closed file. Use the Reset statement so that the operating system flushes its buffers to disk.

Example This example opens a file for random access, gets the contents of one variable, and closes the file again. The subprogram, `CreateFile`, creates the file `c:\temp001` used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   Rem Put the numbers 1-10 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 1 to 10
      Write #1, x
   Next x
   Close #1
   Reset
End Sub
```

```
Sub Button1_Click
   Dim acctno as String * 3
   Dim recno as Long
   Dim msgtext as String
   Call CreateFile
   recno = 1
   newline = Chr(10)
   Open "c:\temp001" For Random As #1 Len = 3
   msgtext = "The account numbers are:" & newline & newline
   Do Until recno = 11
      Get #1,recno,acctno
      msgtext = msgtext & acctno
      recno = recno + 1
   Loop
   Close #1
   Reset
   Kill "c:\temp001"
End Sub
```

See Also

"Open Statement" on page 344
"Reset Statement" on page 385
"Stop Statement" on page 461

Const Statement

This standard VB statement declares symbolic constants for use in a Basic program.

Syntax [Global] Const *constantName* [As *type*] = *expression* [, *constantName* [As *type*] = *expression*] ...

Argument	Description
constantName	The variable name to contain a constant value
type	The data type of the constant (Number or String)
expression	Any expression that evaluates to a constant number

Returns Not applicable

Usage Instead of using the As clause, the type of the constant can be specified by using a type character as a suffix (*#* for numbers, *$* for strings) to *constantName*. If no type character is specified, the type of *constantName* is derived from the type of the expression.

To specify a Global Const, you must follow the same rules as for declaring a Global variable: It must be declared in the (general) (declarations) section of the modules in which you wish to access the Global variable.

Example For an example, read “CLng Function” on page 128.

See Also “Declare Statement” on page 155
“Deftype Statement” on page 158
“Dim Statement” on page 161
“Let (Assignment Statement)” on page 293
“Type Statement” on page 487

Copy Method

Copy returns a copy of a property set. For details, read *Siebel Object Interfaces Reference*.

Cos Function

This standard VB function returns the cosine of an angle.

Syntax Cos(number)

Argument	Description
number	An angle in radians

Returns The cosine of *number.*

Usage The return value is between -1 and 1. The return value is a single-precision number if the angle has a data type of integer or currency, or is a single-precision value. The return value is a double-precision value if the angle has a data type of long or variant, or is a double-precision value.

The angle can be either positive or negative. To convert degrees to radians, multiply by (PI/180). The value of PI is approximately 3.14159.

Example This example finds the length of a roof, given its pitch and the distance of the house from its center to the outside wall.

```
Sub Button_Click
   Dim bwidth As Single, roof As Single, pitch As Single
   Dim msgtext
   Const PI = 3.14159
   Const conversion = PI/180
   pitch = 35
   pitch = Cos(pitch * conversion)
   bwidth = 75
   roof = bwidth/pitch
   msgtext = "The length of the roof is " & _
      Format(roof, "##.##") & " feet."
End Sub
```

See Also “Atn Function” on page 90
“Sin Function” on page 433
“Tan Function” on page 471
“Derived Trigonometric Functions” on page 527

CreateObject Function

Creates a new COM automation object.

Syntax CreateObject(*application.objectname*)

Argument	Description
application	The name of the application
objectname	The name of the object to be used

Returns Not applicable

Usage To create an object, you first must declare an object variable, using Dim, and then Set the variable equal to the new object, as follows:

```
Dim excelObj As Object
Set excelObj = CreateObject("Excel.Application")
```

To refer to a method or property of the newly created object, use the syntax *objectvar.property* or *objectvar.method*, as follows:

```
Dim cellVal as String
cellVal = excelObj.ActiveSheet.Cells(1,1).Value
```

Refer to the documentation provided with your Web Client Automation Server application for correct application and object names. Modal or nonmodal forms cannot be displayed from server-based applications. DLLs instantiated by this function should be Thread-Safe.

CAUTION: When invoking a COM object, a 440 error message may occur if you pass the wrong number, order, or type of parameters to the COM object.

Example This example uses CreateObject to create an Excel worksheet and then edits and saves the worksheet.

```
Sub BtnExcel_Click
   Dim oWorkSheet As Object
   Dim sfileName As String
   Set oWorkSheet = CreateObject("Excel.Sheet")
   If oWorkSheet Is Nothing the
      Exit Sub
   End If

   ' Make Excel visible through the Application object.
   oWorkSheet.Application.Visible = 1
   ' Place some text in the first cell of the sheet
   oWorkSheet.ActiveSheet.Cells(1,1).Value = "Column A, Row 1"
   ' Save the sheet
   sfileName = "C:\demo.xls"
   oWorkSheet.SaveAs (fileName)
   ' Close Excel with the Quit method on the Application object
   oWorkSheet.Application.Quit
   ' Clear the object from memory
   Set oWorkSheet = Nothing
End Sub
```

This example uses CreateObject to create a Word document and then edits and saves the document.

```
Sub BtnWrd_Click
   Dim oWord As Object
   Dim fileName As String
   fileName = "C:\demo.doc"
   Set oWord = CreateObject("Word.Application")
   ' Create a new document
   oWord.Documents.Add
   If oWord Is Nothing then
      Exit Sub
   End If
   ' Make Word visible through the Application object
   oWord.Application.Visible = 1
   ' Add some text
   oWord.Selection.TypeText "This is a demo."
   ' Save the document
   oWord.ActiveDocument.SaveAs (fileName)
   ' Close Word with the Quit method on the Application object
   oWord.Quit
   ' Clear the object from memory
   Set oWord = Nothing
End Sub
```

See Also

"GetObject Function" on page 234
"Is Operator" on page 275
"Me" on page 311
"New Operator" on page 325
"Nothing Function" on page 329
"Object Class" on page 336
"Typeof Function" on page 489

CSng Function

This standard VB function converts an expression to the data type single.

Syntax CSng(*expression*)

Argument	Description
expression	Any expression that can evaluate to a number

Returns The value of *expression* as a single-precision floating-point number.

Usage The *expression* must have a value within the range allowed for the single data type, or an error occurs.

Strings that cannot be converted to an integer result in a Type Mismatch error. Variants containing null result in an Illegal Use of Null error.

Example This example calculates the factorial of a number. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.

```
Sub Button_Click
   Dim number as Integer
   Dim factorial as Double
   Dim msgtext As String
   number = 25
   If number <= 0 then
      Exit Sub
   End If

   factorial = 1
   For x = number to 2 step -1
      factorial = factorial * x
   Next x
      'If number  <= 35, then its factorial is small enough to
      ' be stored as a single-precision number
   If number< 35 then
      factorial = CSng(factorial)
```

```
    End If
    msgtext = "The factorial of " & number & " is " & factorial
End Sub
```

See Also

"CCur Function" on page 115
"CDbl Function" on page 117
"CInt Function" on page 123
"CLng Function" on page 128
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144

CStr Function

This standard VB function converts an expression to the data type string.

Syntax CStr(*expression*)

Argument	Description
expression	Any expression that can evaluate to a number

Returns A string containing the value of *expression*.

Expression	Return value
Date	A string containing a date
Empty	A zero-length string ("")
Error	A string containing `Error`, followed by the error number
Null	A run-time error
Other Numeric	A string containing the number

Example This example uses the string functions to operate on a string that was originally entered as a number.

```
Sub Button_Click
   Dim var1, msgtext as String, code as Integer
   var1 = 77

   msgtext = Cstr(var1)
   msgtext = Left(var1,1)
   code = Asc(msgtext)

   msgtext = "The first digit you entered was," & msgtext
   msgtext = msgtext & ". Its ANSI code is " & code & "."
End Sub
```

See Also

"Asc Function" on page 88
"CCur Function" on page 115
"CDbl Function" on page 117
"Chr Function" on page 121
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CVar Function" on page 143
"CVDate Function" on page 144
"Format Function" on page 202

CurDir Function

This standard VB function returns the default folder (and drive) for the specified drive.

Syntax CurDir[$][(*drive*)]

Argument	Description
drive	The letter of the drive to search

Returns The default drive and folder.

Usage A colon is not required after the drive name. The drive must exist, and must be within the range specified in the LASTDRIVE statement of the `config.sys` file. If a null argument (`""`) is supplied, or if no *drive* is indicated, the path for the default drive is returned.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

To change the current drive, use ChDrive. To change the current folder, use ChDir.

Example This example changes the current folder to C:\Windows, if it is not already the default.

```
Sub Button_Click
   Dim newdir as String
   newdir = "c:\Windows"
   If CurDir  <>  newdir then
      ChDir newdir
   End If
End Sub
```

See Also "ChDir Statement" on page 118
"ChDrive Statement" on page 120
"Dir Function" on page 167
"MkDir Statement" on page 318
"RmDir Statement" on page 390

CurrencyCode Method

CurrencyCode returns the operating currency code associated with the division to which the user's position has been assigned. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

CVar Function

This standard VB function converts an expression to the data type variant.

Syntax CVar(*expression*)

Argument	Description
expression	Any expression that can evaluate to a number

Returns The *expression* as an expression of type variant.

Usage CVar accepts any type of *expression*.

CVar generates the same result as you would get by assigning the expression to a variant variable.

Example This example converts a single variable to a variant variable.

```
Sub Button_Click
   Dim singleAnswer as Single
   Dim variantAnswer as Variant
   singleAnswer = 100.5
   variantAnswer = CVar(singleAnswer )
end Sub
```

See Also

"CCur Function" on page 115
"CDbl Function" on page 117
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144

CVDate Function

This standard VB function converts an expression to the data type variant of type date.

Syntax CVDate(*expression*)

Argument	Description
expression	Any expression that can evaluate to a number

Returns The value of *expression* expressed as a variant of vartype 7 (date).

Usage CVDate accepts both string and numeric values.

The CVDate function returns a variant of vartype 7 (date) that represents a date from January 1, 100, through December 31, 9999. A value of 2 represents January 1, 1900. Times are represented as fractional days.

CVDate converts the time portion of a date expression if one is included as part of the expression, or if the time expression is the only argument. For ways to display the desired result of a date conversion, read "Format Function" on page 202.

Example This example displays the date for one week from the date entered by the user.

```
Sub Button_Click
      Dim str1 as String
   Dim nextweek
   Dim msgtext as String
i:
   str1 = "2/5/2001"
   answer = IsDate(str1)
   If answer = -1 then
      str1 = CVDate(str1)
      nextweek = DateValue(str1) + 7
      msgtext = "One week from the date entered is:
      msgtext = msgtext & "Format(nextweek,"dddddd")
   Else
      Goto i
   End If
End Sub
```

See Also

"Asc Function" on page 88
"CCur Function" on page 115
"CDbl Function" on page 117
"Chr Function" on page 121
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"DateValue Function" on page 151
"Format Function" on page 202
"Val Function" on page 495

Date Function

This standard VB function returns a string representing the current date as determined by the computer's clock.

Syntax Date[$]

Argument	Description
Not applicable	

Returns The current date, as a value of type string.

Usage The Date function returns a ten-character string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example This example displays the date for one week from today's date (the current date on the computer).

```
Sub Button_Click
   Dim nextweek
   nextweek = CVar(Date) + 7
End Sub
```

See Also
"CVDate Function" on page 144
"Date Statement" on page 147
"DateSerial Function" on page 149
"Format Function" on page 202
"Now Function" on page 331
"Time Function" on page 473
"Time Statement" on page 475
"Timer Function" on page 477
"TimeSerial Function" on page 479

Date Statement

This standard VB function sets the computer's date.

Syntax Date[$] = *expression*

Argument	Description
expression	A string in one of the following forms: ■ *mm-dd-yy* ■ *mm-dd-yyyy* ■ *mm/dd/yy* ■ *mm/dd/yyyy*

In the preceding string forms, the placeholders are interpreted as follows:

Placeholder	Meaning
mm	A month expressed as a two-digit number (01 to 12)
dd	A day expressed as a two-digit number (01 to 31)
yy	A year expressed as a two-digit number (00 to 99)
yyyy	A year expressed as a four-digit number (1980 to 2099)

Returns Not applicable

Usage If the dollar sign ($) is omitted, *expression* can be a string containing a valid date, a variant of vartype 7 (date), or a variant of vartype 8 (string).

If *expression* is not already a variant of vartype 7 (date), Date attempts to convert it to a valid date from January 1, 1980, through December 31, 2099. Date uses the Short Date format in the International section of Windows Control Panel to recognize day, month, and year if a string contains three numbers delimited by valid date separators. In addition, Date recognizes month names in either full or abbreviated form.

Example This example changes the computer's date to a date entered by the user.

```
Sub Button_Click
   Dim userdate
   Dim answer
i:
   userdate = "2/5/2001"
   If userdate = "" then
      Exit Sub
   End If
   answer = IsDate(userdate)
   If answer = -1 then
      Date = userdate
   Else
      Goto i
   End If
End Sub
```

See Also "Date Function" on page 146
"Time Function" on page 473
"Time Statement" on page 475

DateSerial Function

This standard VB function returns a date value for the year, month, and day specified.

Syntax DateSerial(*year, month, day*)

Argument	Description
year	An integer representing a year between 100 and 2099 or a numeric expression
month	An integer representing a month between 1 and 12 or a numeric expression
day	An integer representing a day between 1 and 31 or a numeric expression

Returns A variant of vartype 7 (date) that represents a date from January 1, 100, through December 31, 9999, where January 1, 1900, is 2.

Usage A numeric expression can be used for any of the arguments to specify a relative date: a number of days, months, or years before or after a certain date.

Example This example finds the day of the week for November 7 in the year 2009.

```
Sub Button_Click
   Dim checkdate As Variant, daynumber As Variant
   Dim msgtext As String, checkday as Variant
   Const checkyear = 2009
   Const checkmonth = 11
   checkday = 7
   checkdate = DateSerial(checkyear,checkmonth,checkday)
   daynumber = Weekday(checkdate)
   msgtext = "November 7, 2009 falls on a " & _
      Format(daynumber, "dddd")
End Sub
```

See Also

"DateValue Function" on page 151
"Day Function" on page 153
"Format Function" on page 202
"Month Function" on page 320
"Now Function" on page 331
"TimeSerial Function" on page 479
"TimeValue Function" on page 481
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

DateValue Function

This standard VB function returns a date value for the string specified.

Syntax DateValue(*date*)

Argument	Description
date	A string representing a valid date

Returns A variant of vartype 7 (date) that represents a date from January 1, 100, through December 31, 9999, where January 1, 1900, is 2.

Usage DateValue accepts several different string representations for a date. It makes use of the operating system's international settings for resolving purely numeric dates. In contrast to the CVDate function (read "CVDate Function" on page 144), the argument to the DateValue function must be in a valid date format. If given a time in acceptable format, DateValue changes the time to 12:00:00 AM regardless of the value given. If given a number that is not an acceptable date or time format, DateValue returns a Type Mismatch error. For ways to display the desired result of a date conversion, read "Format Function" on page 202.

Example This example displays the date for one week from the date entered by the user.

```
Sub Button_Click
   Dim str1 As String, answer As Integer, msgtext As String
   Dim nextweek
i:
   str1 = "12/22/2000"
   answer = IsDate(str1)
   If answer = -1 then
      str1 = CVDate(str1)
      nextweek = DateValue(str1) + 7
      msgtext = "One week from your date is: "
      msgtext = msgtxt & Format(nextweek,"dddddd")
   Else
      msgtext = "Invalid date or format. Try again."
      Goto i
   End If
End Sub
```

See Also

"DateSerial Function" on page 149
"Day Function" on page 153
"Format Function" on page 202
"Month Function" on page 320
"Now Function" on page 331
"TimeSerial Function" on page 479
"TimeValue Function" on page 481
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

Day Function

This standard VB function returns the day of the month of a date-time value.

Syntax Day(*date*)

Argument	Description
date	Any expression that can evaluate to a date

Usage Day attempts to convert the input value of *date* to a date value. The return value is a variant of vartype 2 (integer). If the value of *date* is null, a variant of vartype 1 (null) is returned.

Example This example finds the month (1 to 12) and day (1 to 31) values for this Thursday.

```
Sub Button_Click
   Dim x As Integer, Today As Variant, msgtext As String
   Today = DateValue(Now)
   Let x = 0
   Do While Weekday(Today + x) <> 5
   x = x + 1
   Loop
   msgtext = "This Thursday is: " & Month(Today + x) & "/" & _
      Day(Today + x)
End Sub
```

See Also

"Date Function" on page 146
"Date Statement" on page 147
"Hour Function" on page 257
"Minute Function" on page 316
"Month Function" on page 320
"Now Function" on page 331
"Second Function" on page 397
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

DeactivateFields Method

DeactivateFields deactivates the Fields that are currently active from a business component SQL query statement. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Declare Statement

This standard VB statement declares a procedure in a module or dynamic link library (DLL).

Syntax A Declare Sub *name* [(*parameter* [As *type*])]

Syntax B Declare Function *name* [(*parameter* [As *type*])] [As *funcType*]

Argument	Description
name	The name of the subprogram or function procedure to declare
parameter	The arguments to pass to the procedure, separated by commas
type	The data type for the arguments
funcType	The data type of the return value of the function

Returns **Syntax A:** Not applicable

Syntax B: A value of the type *funcType,* which can be used in an expression.

Usage To specify the data type for the return value of a function, end the function name with a type character or use the As *funcType* clause shown previously. If no type is provided, the function defaults to data type variant.

Siebel Tools compiles custom functions in alphabetical order. Therefore, when a procedure in the current module is referenced before it is defined, a declaration *must* be used.

For example, suppose you have created the following subroutines in the (general) (declarations) section of a module:

```
Sub A
' Calling B
B
End Sub
```

```
Sub B
theApplication.RaiseErrorText "Sub B called"
End Sub
```

Compilation fails with the message, “Unknown function: B.” However, place the statement:

```
Declare Sub B
```

before Sub A and the code compiles and runs properly.

The data type of a parameter can be specified by using a type character or by using the As clause. Record parameters are declared by using an As clause and a *type* that has previously been defined using the Type statement. Array parameters are indicated by using empty parentheses after the *parameter*; array dimensions are not specified in the Declare statement.

External DLL procedures are called with the Pascal calling convention (the actual arguments are pushed on the stack from left to right). By default, the actual arguments are passed by Far reference. For external DLL procedures, there are two additional keywords, ByVal and Any, that can be used in the parameter list.

When ByVal is used, it must be specified before the parameter it modifies. When applied to numeric data types, ByVal indicates that the parameter is passed by value, not by reference. When applied to string parameters, ByVal indicates that the string is passed by Far pointer to the string data. By default, strings are passed by Far pointer to a string descriptor.

Any can be used as a type specification, and permits a call to the procedure to pass a value of any datatype. When Any is used, type checking on the actual argument used in calls to the procedure is disabled (although other arguments not declared as type Any are fully type-safe). The actual argument is passed by Far reference, unless ByVal is specified, in which case the actual value is placed on the stack (or a pointer to the string in the case of string data). ByVal can also be used in the call. The external DLL procedure has the responsibility of determining the type and size of the passed-in value.

When an empty string ("") is passed ByVal to an external procedure, the external procedure receives a valid (non-NULL) pointer to a character of 0. To send a NULL pointer, Declare the procedure argument as ByVal As Any, and call the procedure with an argument of 0.

Example This example declares a function that is later called by the main subprogram. The function does nothing but set its return value to 1. For other examples of functions, read "Function...End Function Statement" on page 213 and "GoTo Statement" on page 253.

```
(general) (declarations)
Option Explicit
Declare Function SVB_exfunction()

Function SVB_exfunction()
   SVB_exfunction = 1
End Function

Sub Button_Click
   Dim y as Integer
   Call SVB_exfunction
   y = SVB_exfunction
End Sub
```

See Also

"Const Statement" on page 131
"Deftype Statement" on page 158
"Dim Statement" on page 161
"Static Statement" on page 460
"Type Statement" on page 487

Deftype Statement

This standard VB statement specifies the default data type for one or more variables.

Syntax

DefCur *varTypeLetters*
DefInt *varTypeLetters*
DefLng *varTypeLetters*
DefSng *varTypeLetters*
DefDbl *varTypeLetters*
DefStr *varTypeLetters*
DefVar *varTypeLetters*

Argument	Description
varTypeLetter	The first letter of a variable name to use

Returns

Not applicable

Usage

VarTypeLetters can be a single letter, a comma-separated list of letters, or a range of letters. For example, a-d indicates the letters a, b, c, and d.

The case of the letters is not important, even in a letter range. The letter range a-z is treated as a special case: it denotes all alpha characters, including the international characters.

The Def*type* statement affects only the module in which it is specified. It must precede any variable definition within the module.

Variables defined using a Global or Dim statement can override the Def*type* statement by using an As clause or a type character.

Example

This example finds the average of bowling scores entered by the user. Because the variable *average* begins with A, it is defined as a single-precision floating point number. The other variables are defined as integers.

```
DefInt c,s,t
DefSng a
Sub Button_Click
```

```
      Dim count
      Dim total
      Dim score
      Dim average
      Dim msgtext
      For count = 0 to 4
         score = 180
         total = total + score
      Next count
      average = total/count
      msgtext = "Your average is: " &average
   End Sub
```

See Also

“Declare Statement” on page 155
“Dim Statement” on page 161
“Global Statement” on page 249
“Let (Assignment Statement)” on page 293
“Type Statement” on page 487

DeleteRecord Method

DeleteRecord removes the current record from a Siebel business component. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Dim Statement

This standard VB statement declares variables for use in a Basic program.

Syntax Dim [Shared] *variableName* [As[New] *type*] [, *variableName* [As[New] *type*]] ...

Placeholder	Description
variableName	The name of the variable to declare
type	The data type of the variable

Returns Not applicable

Usage Dim is a declaration statement. It is an abbreviation for Declare in Memory; however, you must use the short form.

VariableName must begin with a letter and contain only letters, numbers, and underscores. A name can also be delimited by brackets, and any character can be used inside the brackets, except for other brackets.

```
Dim my_1st_variable As String

Dim [one long and strange! variable name] As String
```

If the As clause is not used, the *type* of the variable can be specified by using a type character as a suffix to *variableName*. The two different type-specification methods can be intermixed in a single Dim statement (although not on the same variable).

Basic is a strongly typed language: variables must be given a data type or they are assigned the data type variant. The available data types are:

- Array
- Double (double-precision floating-point number)
- Integer
- Long (double-precision integer)
- Object

- Record
- Single (single-precision floating-point number)
- String
- Variant

For details on these variable types, read "Data Types" on page 54.

NOTE: Good programming practice is to declare every variable. To force variables to be explicitly declared, use the Option Explicit statement (read "Option Explicit Statement" on page 352). Place procedure-level Dim statements at the beginning of the procedure.

Variables can be shared across modules. A variable declared inside a procedure has scope local to that procedure. A variable declared outside a procedure has scope local to the module. If you declare a variable with the same name as a module variable, the module variable is not accessible. For details, read "Global Statement" on page 249.

The Shared keyword is included for backward compatibility with older versions of Basic. It is not allowed in Dim statements inside a procedure. It has no effect.

Regardless of which mechanism you use to declare a variable, you can choose to use or omit the type character when referring to the variable in the rest of your program. The type suffix is not considered part of the variable name.

CAUTION: You can declare several variables on one line; however, unless you include the type for each variable, the type applies only to the last variable declared.

For example,

```
Dim Acct, CustName, Addr As String
```

causes only `Addr` to be declared as type string; the other variables are implicitly declared as type variant. On the other hand,

```
Dim Acct As String, CustName As String, Addr As String
```

declares the variables as type string.

Arrays

The available data types for arrays are numbers, strings, variants, and records. Arrays of arrays and objects are not supported.

Array variables are declared by including a subscript list as part of the *variableName*. The syntax to use for *variableName* is:

```
Dim variable([[startSubcript To] endSubscript, ...]) As typeName
```

or

```
Dim variable_with_suffix([[startSubcript To] endSubscript, ... ])
```

Argument	Description
startSubscript	[Optional] the index number of the first array element, followed by the keyword To
endSubscript	The index number of the last element of the array

If *startSubscript* is not specified, 0 is used as the default. Thus, the statement
`Dim counter (25) as Integer`
creates an array named `counter` that has 26 elements (0 through 25). To change the default, use the Option Base statement.

Both *startSubscript* and *endSubscript* are valid subscripts for the array. The maximum number of subscripts that can be specified in an array definition is 60. The maximum total size for an array is limited only by the amount of memory available.

If no *subscriptRange* is specified for an array, the array is declared as a dynamic array. In this case, the ReDim statement must be used to specify the dimensions of the array before the array can be used.

Numbers

Numeric variables can be declared using the As clause and one of the following numeric types: currency, integer, long, single, double. Numeric variables can also be declared by including a type character as a suffix to the name. Numeric variables are initialized to 0.

Objects

Object variables are declared using an As clause and a *typeName* of a class. Object variables can be set to refer to an object, and then used to access members and methods of the object using dot notation.

```
Dim COMObject As Object
Set COMObject = CreateObject("spoly.cpoly")
COMObject.reset
```

An object can be declared as New for some classes. For example:

```
Dim variableName As New className
variableName.methodName
```

In such instances, a Set statement is not required to create the object variable; a new object is allocated when the variable is used.

NOTE: The New operator cannot be used with the Basic Object class.

Records

Record variables are declared by using an As clause and a *typeName* that has been defined previously using the Type statement. The syntax to use is:

```
Dim variableName As typeName
```

Records are made up of a collection of data elements called fields. These fields can be of any numeric, string, variant, or previously defined record type. For details on accessing fields within a record, read "Type Statement" on page 487.

Strings

Siebel VB supports two types of strings: fixed-length and dynamic. Fixed-length strings are declared with a specific length (between 1 and 32767) and cannot be changed later. Use the following syntax to declare a fixed-length string:

```
Dim variableName As String * length
```

Dynamic strings have no declared length, and can vary in length from 0 to 32,767. The initial length for a dynamic string is 0. Use the following syntax to declare a dynamic string:

```
Dim variableName$
```

or

```
Dim variableName As String
```

When initialized, fixed-length strings are filled with zeros. Dynamic strings are initialized as zero-length strings.

Variants

Declare variables as variants when the type of the variable is not known at the start of, or might change during, the procedure. For example, a variant is useful for holding input from a user when valid input can be either text or numbers. Use the following syntax to declare a variant:

```
Dim variableName
```

or

```
Dim variableName As Variant
```

Variant variables are initialized to vartype Empty.

Example This example shows a Dim statement for each of the possible data types.

```
' Must define a record type before you can declare a record
' variable
   Type Testrecord
      Custno As Integer
      Custname As String
   End Type

Sub Button_Click
   Dim counter As Integer
   Dim fixedstring As String * 25
   Dim varstring As String
   Dim myrecord As Testrecord
   Dim ole2var As Object
   Dim F(1 to 10), A()
   '...(code here)...
End Sub
```

See Also

"Global Statement" on page 249
"Option Base Statement" on page 347
"ReDim Statement" on page 377
"Service_InvokeMethod Event" on page 409
"Static Statement" on page 460
"Type Statement" on page 487

Dir Function

The standard VB function Dir returns a filename that matches the specified pattern, having the specified attributes.

Syntax Dir[$] [(*pathname*[, *attributes*])]

Argument	Description
pathname	A string or string expression evaluating to a path or filename
attributes	An integer expression specifying the file attributes to select

Returns The first filename that matches the *pathname* argument and has the specified attributes. Use the following integer values for *attributes* to return the specified type of file.

Integer	File Type
0 (default)	Normal files (no attributes set)
2	Normal and hidden files
4	Normal and system files
8	Volume label (only)
16	Normal files and folders

NOTE: The values in the table can be added together to select multiple attributes. For example, to list hidden and system files in addition to normal files, set *attributes* to 6 (6 = 2 + 4). If *attributes* is set to 8, the Dir function returns the volume label of the drive specified in the *pathname* argument, or of the current drive if a drive is not explicitly specified.

Usage *Pathname* can include a drive specification and wildcard characters (? and *). An empty string ("") passed as *pathname* is interpreted as the current folder (the same as "."). To retrieve additional matching filenames, call the Dir function again, omitting the *pathname* and *attributes* arguments. If no file is found, an empty string ("") is returned.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise, the function returns a variant of vartype 8 (string).

Example This example lists the contents of the diskette in drive A.

```
Sub Button_Click
   Dim msgReturn
   Dim folder, count
   Dim x, msgtext
   Dim A()
   count = 1

   ReDim A(100)
   folder = Dir ("A:\*.*")
   Do While folder <> ""
      A(count) = folder
      Count = count + 1
      folder = Dir
   loop
   msgtext = "Contents of drive A:\ is:" & Chr(10) & Chr(10)
   For x = 1 to count
      msgtext = msgtext & A(x) & Chr(10)
   Next x
End Sub
```

See Also "ChDir Statement" on page 118
"ChDrive Statement" on page 120
"CurDir Function" on page 141
"MkDir Statement" on page 318
"RmDir Statement" on page 390

Do...Loop Statement

This standard VB control structure repeats a series of program lines as long as (or until) an expression is TRUE.

Syntax A

Do [{ While|Until } *condition*]

statement_block

[Exit Do]

statement_block

Loop

Syntax B

Do

statement_block

[Exit Do]

statement_block

Loop [{ While|Until } *condition*]

Placeholder	Description
condition	Any expression that evaluates to TRUE (non-zero) or FALSE (0)
statement_block	Program lines to repeat while (or until) *condition* is TRUE (non-zero)

Returns

Not applicable

Usage

When an Exit Do statement is executed, control goes to the statement after the Loop statement. When used within a nested loop, an Exit Do statement moves control out of the immediately enclosing loop.

Example

For examples, read "Dir Function" on page 167, "Eof Function" on page 173, and "Err Function" on page 179.

See Also

"Exit Statement" on page 187
"Stop Statement" on page 461
"While...Wend Statement" on page 507

Environ Function

This standard VB function returns the string setting for a keyword in the operating system's environment table.

Syntax A Environ[$](*environment-string*)

Syntax B Environ[$](*numeric_expression*)

Argument	Description
environment-string	The name of a keyword in the operating system
numeric_expression	An integer for the position of the string in the environment table (1st, 2nd, 3rd, and so on)

Returns The string value assigned to an environment variable.

Usage If you use the *environment-string* parameter, enter it in uppercase, or Environ returns a null string (""). The return value for Syntax A is the string associated with the keyword requested.

If you use the *numeric_expression* parameter, the numeric expression is rounded to a whole number, if necessary. The return value for Syntax B is a string in the form `KEYWORD=value`.

Environ returns a null string if the specified argument cannot be found.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example This example lists the strings from the operating system environment table.

```
Sub Button_Click
   Dim str1(100)
   Dim msgtext
   Dim count, x
   Dim newline
   newline = Chr(10)
   x = 1
```

```
      str1(x) = Environ(x)
      Do While Environ(x) <> ""
         str1(x) = Environ(x)
         x = x + 1
         str1(x) = Environ(x)
      Loop
      msgtext = "The Environment Strings are:" & newline & newline
      count = x
      For x = 1 to count
         msgtext = msgtext & str1(x) & newline
      Next x
   End Sub
```

Eof Function

This standard VB function is used to determine whether the end of an open file has been reached.

Syntax

Eof(*filenumber*)

Argument	Description
filenumber	The file number used in the Open statement to open the file

Returns

The value -1 if the end of the specified open file has been reached, 0 otherwise.

Usage

For more information about assigning numbers to files when they are opened, read "Open Statement" on page 344.

Example

This example uses the Eof function to read records from a Random file, using a Get statement. The Eof function keeps the Get statement from attempting to read beyond the end of the file. The subprogram, `CreateFile`, creates the file `C:\TEMP001` used by the main subprogram. For another example, read "FileDateTime Function" on page 194.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   ' Put the numbers 1-10 into a file
   Dim x as Integer
   Open "C:\TEMP001" for Output as #1
   For x = 1 to 10
      Write #1, x
   Next x
   Close #1
End Sub

Sub Button_Click
   Dim acctno
   Dim msgtext as String
   newline = Chr(10)
   Call CreateFile
```

```
        Open "C:\temp001" For Input As #1
        msgtext = "The account numbers are:" & newline
        Do While Not Eof(1)
              Input #1,acctno
              msgtext = msgtext & newline & acctno & newline
        Loop
        Close #1
        Kill "C:\TEMP001"
    End Sub
```

See Also

"Get Statement" on page 218
"Input Function" on page 261
"Input Statement" on page 262
"Line Input Statement" on page 297
"Loc Function" on page 299
"Lof Function" on page 302
"Open Statement" on page 344

Erase Statement

This standard VB statement reinitializes the contents of a fixed array or frees the storage associated with a dynamic array.

Syntax

Erase *Array*[, *Array*]

Argument	Description
Array	The name of the array variable to re-initialize

Returns

Not applicable

Usage

The effect of using Erase on the elements of a fixed array varies with the type of the element:

Element Type	Erase Effect
Numeric	Each element is set to zero.
Variable-length string	Each element is set to a zero-length string ("").
Fixed-length string	Each element's string is filled with zeros.
Variant	Each element is set to Empty.
User-defined type	Members of each element are cleared as if the members were array elements; that is, numeric members have their values set to zero, the strings to "", and so on.
Object	Each element is set to the special value Nothing.

Example

This example prompts for a list of item numbers to put into an array and clears the array if the user wants to start over.

```
Sub Button_Click
   Dim msgtext
   Dim inum(100) as Integer
   Dim x, count
   Dim newline
   newline = Chr(10)
```

```
    x = 1
    count = x
    inum(x) = 0
    Do
       inum(x) = x + 1
       If inum(x) = 99 then
          Erase inum()
          x = 0
       ElseIf inum(x) = 0 then
          Exit Do
       End If
       x = x + 1
    Loop
    count = x-1
    msgtext = "You entered the following numbers:" & newline
    For x = 1 to count
       TheApplication.TraceOn "c:\temp\trace.txt", "Allocation", 
"All"
       TheApplication.Trace msgtext & inum(x) & newline
       Next x
End Sub
```

See Also

"Dim Statement" on page 161
"LBound Function" on page 287
"ReDim Statement" on page 377
"UBound Function" on page 490

Erl Function

This standard VB function returns the line number where an error was trapped.

Syntax Erl

Argument	Description
Not applicable	

Returns The line number on which an error occurred.

Usage If you use a Resume or On Error statement after Erl, the return value for Erl is reset to 0. To maintain the value of the line number returned by Erl, assign it to a variable.

The value of the Erl function can be set indirectly through the Error statement.

Example This example prints the error number using the Err function and the line number using the Erl statement if an error occurs during an attempt to open a file. Siebel VB assigns line numbers, starting with 1, which is the `Sub Button_Click` statement.

```
Sub Button_Click
   Dim msgtext, userfile
   On Error GoTo Debugger
   msgtext = "Enter the filename to use:"
   userfile = "c:\temp\trace.txt"
   Open userfile For Input As #1
   ' ....etc....
   Close #1
done:
   Exit Sub

Debugger:
   msgtext = "Error number " & Err & " occurred at line: " & Erl
      Resume done
End Sub
```

See Also

"Err Function" on page 179
"Err Statement" on page 180
"Error Function" on page 182
"Error Statement" on page 184
"On Error Statement" on page 341
"Resume Statement" on page 387
"Trappable Errors" on page 523

Err Function

This standard VB function returns the run-time error code for the last error trapped.

Syntax Err

Argument	Description
Not applicable	

Returns The run-time error code for the last standard VB error trapped.

Usage If you use a Resume or On Error statement after Erl, the return value for Err is reset to 0. To maintain the value of the line number returned by Erl, assign it to a variable.

The value of the Err function can be set directly through the Err statement, and indirectly through the Error statement.

The standard VB trappable errors are listed in "Trappable Errors" on page 523

CAUTION: You cannot view Siebel VB errors with this function. Instead, use the appropriate method for the Siebel interface you are using (COM, ActiveX, or CORBA). Error trapping methods and examples for each interface are documented in *Siebel Object Interfaces Reference*.

Example For examples, read "Erl Function" on page 177 and "Error Function" on page 182.

See Also
"Erl Function" on page 177
"Err Statement" on page 180
"Error Function" on page 182
"Error Statement" on page 184
"On Error Statement" on page 341
"Resume Statement" on page 387
"Trappable Errors" on page 523

Err Statement

This standard VB statement sets a run-time error code.

Syntax Err = *errornumber*

Argument	Description
errornumber	An integer between 1 and 32,767 representing an error code, or a 0 if no error occurs

Returns Not applicable

Usage The Err statement is used to send error information between procedures.

Example This example generates an error code of 10000 and displays an error message if a user does not enter a customer name when prompted for it. It uses the Err statement to clear any previous error codes before running the loop the first time, and it also clears the error to allow the user to try again. For another example, read "Error Statement" on page 184.

```
Sub Button_Click
   Dim custname as String
   On Error Resume Next
   Do
      Err = 0
      custname = "Acme Inc."
      If custname = "" then
         Error 10000
      Else
         Exit Do
      End If
      Select Case Err
         Case 10000
            TheApplication.RaiseErrorText "You must enter a 
customer name."
         Case Else
            TheApplication.RaiseErrorText "Undetermined error. 
Try again."
      End Select
```

```
    Loop Until custname <> ""
       TheApplication.RaiseErrorText "The name is: " & custname
End Sub
```

See Also

"Erl Function" on page 177
"Err Function" on page 179
"Error Function" on page 182
"Error Statement" on page 184
"On Error Statement" on page 341
"Resume Statement" on page 387
"Trappable Errors" on page 523

Error Function

This standard VB function returns the error message that corresponds to the specified error code.

Syntax Error[$] [(*errornumber*)]

Argument	Description
errornumber	An integer between 1 and 32,767 representing an error code

Returns The text of the error message corresponding to the error code; if this argument is omitted, Siebel VB returns the error message for the run-time error that has occurred most recently.

If no error message is found to match the error code, a null string ("") is returned.

Usage The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

The standard VB trappable errors are listed in "Trappable Errors" on page 523

Example This example prints the error number, using the Err function, and the text of the error, using the Error$ function, if an error occurs during an attempt to open a file.

```
Sub Button_Click
   Dim msgtext, userfile
   On Error GoTo Debugger
   msgtext = "Enter the filename to use:"
   userfile = "c:\temp\trace.txt"
   Open userfile For Input As #1
   ' ....etc....
   Close #1
done:
   Exit Sub
Debugger:
   msgtext = "Error " & Err & ": " & Error$
      Resume done
End Sub
```

See Also

"Erl Function" on page 177
"Err Function" on page 179
"Err Statement" on page 180
"Error Statement" on page 184
"On Error Statement" on page 341
"Resume Statement" on page 387
"Trappable Errors" on page 523

Error Statement

This standard VB statement simulates the occurrence of a Siebel VB or user-defined error.

Syntax

Error *errornumber*

Argument	Description
errornumber	An integer between 1 and 32,767 representing an error code

Usage

If an *errornumber* is one that Siebel VB already uses, the Error statement simulates an occurrence of that error.

User-defined error codes should employ values greater than those used for standard Siebel VB error codes. To help make sure that non-Siebel VB error codes are chosen, user-defined codes should work down from 32,767.

CAUTION: Error codes for the Siebel VB methods described in *Siebel Object Interfaces Reference* are between 4000 and 4999. Do not use codes in this range for user-defined error codes.

If an Error statement is executed, and there is no error-handling routine enabled, Siebel VB produces an error message and halts program execution. If an Error statement specifies an error code not used by Siebel VB, the message “User-defined error” is displayed.

See Also

“Erl Function” on page 177
“Err Function” on page 179
“Err Statement” on page 180
“Error Function” on page 182
“Error Statement” on page 184
“On Error Statement” on page 341
“Resume Statement” on page 387
“Trappable Errors” on page 523

ExecuteQuery Method

ExecuteQuery returns a set of business component records using the criteria established with methods such as SetSearchSpec. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

ExecuteQuery2 Method

ExecuteQuery2 returns a set of business component records using the criteria established with methods such as SetSearchSpec. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Exit Statement

This standard VB statement terminates loop statements or transfers control to a calling procedure.

Syntax Exit {Do | For | Function | Sub}

Returns Not applicable

Usage Use Exit Do inside a Do...Loop statement. Use Exit For inside a For...Next statement. When the Exit statement is executed, control transfers to the statement after the Loop or Next statement. When used within a nested loop, an Exit statement moves control out of the immediately enclosing loop.

Use Exit Function inside a Function...End Function procedure. Use Exit Sub inside a Sub...End Sub procedure.

Example This example uses the On Error statement to trap run-time errors. If there is an error, the program execution continues at the label “Debugger.” The example uses the Exit statement to skip over the debugging code when there is no error.

```
Sub Button_Click
   Dim msgtext, userfile
   On Error GoTo Debugger
   msgtext = "Enter the filename to use:"
   userfile = "c:\temp\trace.txt"
   Open userfile For Input As #1
   ' ....etc....
   Close #1
done:
   Exit Sub
Debugger:
   msgtext = "Error " & Err & ": " & Error$
      Resume done
End Sub
```

See Also “Do...Loop Statement” on page 169
“Function...End Function Statement” on page 213
“Sub...End Sub Statement” on page 467

Exp Function

This standard VB function returns the value *e* (the base of natural logarithms) raised to a power.

Syntax Exp(*number*)

Argument	Description
number	The exponent value for *e*

Returns The value of *e* raised to the power *number*.

Usage If the variable to contain the return value has a data type of integer, currency, or single, the return value is a single-precision value. If the variable has a data type of long, variant, or double, the value returned is a double-precision number.

The constant *e* is approximately 2.718282.

Example This example estimates the value of a factorial of a number entered by the user. A factorial (notated with an exclamation mark, !) is the product of a number and each integer between it and the number 1. For example, 5 factorial, or 5!, is the product of 5*4*3*2*1, or the value 120.

```
Sub Button_Click
   Dim x as Single
   Dim msgtext, PI
   Dim factorial as Double
   PI = 3.14159
i: x = 55
   If x< = 0 then
      Exit Sub
   ElseIf x>88 then
      Goto i
   End If
   factorial = Sqr(2 * PI * x) * (x^x/Exp(x))
   msgtext = "The estimated factorial is: " & Format _
      (factorial, "Scientific")
End Sub
```

See Also

"Abs Function" on page 76
"Fix Function" on page 197
"Int Function" on page 268
"Log Function" on page 303
"Rnd Function" on page 392
"Sgn Function" on page 430
"Sqr Function" on page 459

FileAttr Function

This standard VB function returns the file mode or the operating system handle for an open file.

Syntax

FileAttr(*filenumber, returntype*)

Argument	Description
filenumber	The file number used in the Open statement to open the file
returntype	An integer representing the type of information to return

Returns

If returntype is:	Returns:
1	The file mode of the open file, where ■ 1 indicates Input mode ■ 2 indicates Output mode ■ 8 indicates Append mode
2	The operating system handle of the open file

Usage

The argument *filenumber* is the number used in the Open statement to open the file.

Example

This example closes an open file if it is open in input or output mode. If open in append mode, it writes a range of numbers to the file. The second subprogram, `CreateFile`, creates the file and leaves it open.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   Rem Put the numbers 1-10 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 1 to 10
```

```
      Write #1, x
   Next x
End Sub

Sub Button_Click
   Dim filemode as Integer
   Dim attrib as Integer
   Call CreateFile
   attrib = 1
   filemode = FileAttr(1,attrib)
   If filemode = 1 or 2 then
      Close #1
   Else
      For x = 11 to 15
         Write #1, x
      Next x
      Close #1
   End If
      Kill "c:\temp001"
End Sub
```

See Also

"GetAttr Function" on page 222
"Open Statement" on page 344
"SetAttr Statement" on page 413

FileCopy Statement

This standard VB function copies a file.

Syntax FileCopy [*path1*]*source*, [*path2*]*target*

Argument	Description
path1	The path of the file to copy (optional unless *source*$ is not in the current folder)
source	The name, and if necessary, the path, of the file to copy
path2	The path to the folder to which the file should be copied (optional unless the file is to be copied to the current folder)
target	The name to which the file should be copied

Returns Not applicable

Usage Wildcards (* and ?) are not allowed in any of the arguments. The *source* file cannot be copied if it is opened by Siebel VB for anything other than Read access, or if it is open in another program.

Example This example copies one file to another. Both filenames are specified by the user.

```
Sub Button_Click
   Dim oldfile, newfile
   On Error Resume Next
   oldfile = "c:\temp\trace.txt"
   newfile = "c:\temp\newtrace.txt"
   FileCopy oldfile,newfile
   If Err <> 0 then
      msgtext = "Error during copy. Rerun program."
   Else
      msgtext = "Copy successful."
      End If
End Sub
```

See Also

"FileAttr Function" on page 190
"FileDateTime Function" on page 194
"GetAttr Function" on page 222
"Kill Statement" on page 284
"Name Statement" on page 323

FileDateTime Function

This standard VB function returns the last modification date and time for the specified file.

Syntax FileDateTime(*pathname*)

Argument	Description
pathname	A string or string expression evaluating to the name of the file to query

Returns The date and time the file was last modified.

Usage *Pathname* can contain path and disk information, but cannot include wildcards (* and ?).

See Also "FileLen Function" on page 195
"GetAttr Function" on page 222

FileLen Function

This standard VB function returns the length of the specified file.

Syntax FileLen(*pathname*)

Argument	Description
pathname	A string or string expression evaluating to the name of the file to query

Returns The length of the file specified in *pathname*.

Usage *Pathname* can contain path and disk information, but cannot include wildcards (* and ?).

If the specified file is open, FileLen returns the length of the file before it was opened.

Example This example returns the length of a file.

```
Sub Button_Click
   Dim length as Long
   Dim userfile as String
   Dim msgtext
   On Error Resume Next
   msgtext = "Enter a filename:"
   userfile = "trace.txt"
   length = FileLen(userfile)
   If Err <> 0 then
      msgtext = "Error occurred. Rerun program."
   Else
      msgtext = "The length of " & userfile & " is: " & length
      End If
End Sub
```

See Also "FileDateTime Function" on page 194
"GetAttr Function" on page 222
"Lof Function" on page 302

FirstRecord Method

FirstRecord moves to the first record in a Siebel business component, invoking any associated Basic events. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Fix Function

This standard VB function returns the integer part of a number.

Syntax Fix(*number*)

Argument	Description
number	Any valid numeric expression

Returns The integer part of *number.*

Usage The return value's data type matches the type of the numeric expression. This includes variant expressions, unless the numeric expression is a string (vartype 8) that evaluates to a number, in which case the data type for its return value is vartype 5 (double). If the numeric expression is vartype 0 (empty), the data type for the return value is vartype 3 (long).

For both positive and negative numbers, Fix removes the fractional part of the expression and returns the integer part only. For example, Fix (6.2) returns 6; Fix (-6.2) returns -6.

The effect of this function is the same as that of the Int function, except in the handling of negative numbers. Thus:

- Fix(-8.347) = -8
- Int(-8.347) = -9

Example This example returns the integer portion of a number provided by the user.

```
Sub Button_Click
   Dim usernum
   Dim intvalue
   usernum = 77.54
   intvalue = Fix(usernum)
End Sub
```

See Also

"Abs Function" on page 76
"CInt Function" on page 123
"Exp Function" on page 188
"Int Function" on page 268
"Log Function" on page 303
"Rnd Function" on page 392
"Sgn Function" on page 430
"Sqr Function" on page 459

For...Next Statement

This standard VB control structure repeats a series of program lines a fixed number of times.

Syntax For *counter* = *start* To *end* [Step *increment*]

statement_block

[Exit For]

statement_block

Next [*counter*]

Placeholder	Description
counter	A numeric variable for the loop counter
start	The initial value of the counter
end	The ending value of the counter
increment	The amount by which the counter is changed each time the loop is run; the default is 1
statement_block	the Basic functions, statements, or methods to be executed

Returns Not applicable

Usage The *start* and *end* values must be consistent with *increment*. If *end* is greater than *start*, *increment* must be positive. If *end* is less than *start*, *increment* must be negative. Siebel VB compares the sign of (*start* - *end*) with the sign of *increment*. If the signs are the same, and *end* does not equal *start*, the For...Next loop is started. If not, the loop is omitted in its entirety.

With a For...Next loop, the program lines following the For statement are executed until the Next statement is encountered. At this point, the Step amount is added to the counter and compared with the final value, *end*. If the beginning and ending values are the same, the loop executes once, regardless of the Step value. Otherwise, the Step value controls the loop as follows:

Step Value	Loop Execution
Positive	If *counter* is less than or equal to *end*, the Step value is added to *counter*. Control returns to the statement after the For statement and the process repeats. If *counter* is greater than *end*, the loop is exited; execution resumes with the statement following the Next statement.
Negative	The loop repeats until *counter* is less than *end*.
Zero	The loop repeats indefinitely.

Within the loop, the value of the *counter* should not be changed, as changing the *counter* makes programs more difficult to maintain and debug.

For...Next loops can be nested within one another. Each nested loop should be given a unique variable name as its *counter*. The Next statement for the inside loop must appear before the Next statement for the outside loop. The Exit For statement can be used as an alternative exit from For...Next loops.

If the variable is left out of a Next statement, the Next statement matches the most recent For statement. If a Next statement occurs prior to its corresponding For statement, Siebel VB returns an error message.

Multiple consecutive Next statements can be merged together. If this is done, the counters must appear with the innermost counter first and the outermost counter last. For example:

```
For i = 1 To 10
   statement_block
   For j = 1 To 5
      statement_block
Next j, i
```

Example For an example, read "CSng Function" on page 137.

See Also "Do...Loop Statement" on page 169
"Exit Statement" on page 187
"While...Wend Statement" on page 507

Format Function

This standard VB function returns a formatted string of an expression based on a given format.

Syntax Format[$](*expression*[, *format*])

Argument	Description
expression	The value to be formatted; it can be a number, string, or variant
format	A string expression representing the format to use

Select one of the topics that follow for a detailed description of format strings.

Returns The *expression* in the specified *format*.

Usage The Format function formats the *expression* as a number, date, time, or string depending upon the *format* argument. The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string). As with any string, you must enclose the *format* argument in quotation marks ("").

Numeric values are formatted as either numbers or date/times. If a numeric expression is supplied and the *format* argument is omitted or null, the number is converted to a string without any special formatting.

Both numeric values and variants can be formatted as dates. When formatting numeric values as dates, the value is interpreted according the standard Basic date encoding scheme. The base date, December 30, 1899, is represented as zero, and other dates are represented as the number of days from the base date.

Strings are formatted by transferring one character at a time from the input *expression* to the output string.

CAUTION: The Format function does not give the correct format if the *format* string does not match the Regional Settings, or if the Date in the Windows setting is not set to the U.S. format.

Formatting Numbers

The predefined numeric formats with their meanings are as follows:

Format	Description
General Number	Displays the number without thousand separator.
Fixed	Displays the number with at least one digit to the left and at least two digits to the right of the decimal separator.
Standard	Displays the number with thousand separator and two digits to the right of decimal separator.
Scientific	Displays the number using standard scientific notation.
Currency	Displays the number using a currency symbol as defined in the International section of the Control Panel. Use thousand separator and display two digits to the right of decimal separator. Enclose negative value in parentheses.
Percent	Multiplies the number by 100 and displays it with a percent sign appended to the right; displays two digits to the right of decimal separator.
True/False	Displays FALSE for 0, TRUE for any other number.
Yes/No	Displays No for 0, Yes for any other number.
On/Off	Displays Off for 0, On for any other number.

To create a user-defined numeric format, follow these guidelines:

For a simple numeric format, use one or more digit characters and (optionally) a decimal separator. The two format digit characters provided are zero, “0”, and number sign, “#”. A zero forces a corresponding digit to appear in the output; while a number sign causes a digit to appear in the output if it is significant (in the middle of the number or non-zero).

Number	Format	Result
1234.56	#	1235
1234.56	#.##	1234.56

Number	Format	Result
1234.56	#.#	1234.6
1234.56	######.##	1234.56
1234.56	00000.000	01234.560
0.12345	#.##	.12
0.12345	0.##	0.12

A comma placed between digit characters in a format causes a comma to be placed between every three digits to the left of the decimal separator.

Number	Format	Result
1234567.8901	#,#.##	1,234,567.89
1234567.8901	#,#.####	1,234,567.8901

NOTE: Although a comma and period are used in the format specification to denote separators for thousands and decimals, the output string contains the appropriate character based upon the current international settings for your machine.

Numbers can be scaled either by inserting one or more commas before the decimal separator or by including a percent sign in the format specification. Each comma preceding the decimal separator (or after all digits if no decimal separator is supplied) scales (divides) the number by 1000. The commas do not appear in the output string. The percent sign causes the number to be multiplied by 100. The percent sign appears in the output string in the same position as it appears in *format*.

Number	Format	Result
1234567.8901	#,.##	1234.57
1234567.8901	#,,.####	1.2346

Number	Format	Result
1234567.8901	#,#,.##	1,234.57
0.1234	#0.00%	12.34%

Characters can be inserted into the output string by being included in the format specification. The following characters are inserted in the output string in a location matching their position in the format specification:

- \+ $ (space

Any set of characters can be inserted by enclosing them in double quotes. Any single character can be inserted by preceding it with a backslash, \.

Number	Format	Result
1234567.89	$#,0.00	$1,234,567.89
1234567.89	"TOTAL:" $#,#.00	TOTAL: $1,234,567.89
1234	\ = \>#,#\<\ =	= >1,234< =

You can use the standard VB Chr function if you need to embed quotation marks in a format specification. The character code for a quotation mark is 34.

Numbers can be formatted in scientific notation by including one of the following exponent strings in the *format* specification:

E- E+ e- e+

Precede the exponent string with one or more digit characters. The number of digit characters following the exponent string determines the number of exponent digits in the output. Format specifications containing an uppercase `E` result in an uppercase `E` in the output. Those containing a lowercase `e` result in a lowercase `e` in the output. A minus sign following the `E` causes negative exponents in the output to be preceded by a minus sign. A plus sign in the format causes a sign to always precede the exponent in the output.

Number	Format	Result
1234567.89	`###.##E-00`	123.46E04
1234567.89	`###.##e + #`	123.46e + 4
0.12345	`0.00E-00`	1.23E-01

A numeric format can have up to four sections, separated by semicolons. If you use only one section, it applies to every value. If you use two sections, the first section applies to positive values and zeros, the second to negative values. If you use three sections, the first applies to positive values, the second to negative values, and the third to zeros. If you include semicolons with nothing between them, the undefined section is printed using the format of the first section. The fourth section applies to Null values. If it is omitted and the input expression results in a NULL value, Format returns an empty string.

Number	Format	Result
1234567.89	`#,0.00;(#,0.00);"Zero";"NA"`	1,234,567.89
-1234567.89	`#,0.00;(#,0.00);"Zero";"NA"`	(1,234,567.89)
0.0	`#,0.00;(#,0.00);"Zero";"NA#"`	Zero
0.0	`#,0.00;(#,0.00);;"NA"`	0.00
Null	`#,0.00;(#,0.00);"Zero";"NA"`	NA
Null	`"The value is: "`	0.00

Formatting Dates and Times

As with numeric formats, there are several predefined formats for formatting dates and times:

Format	Description
General Date	If the number has both integer and real parts, displays both date and time (for example, 11/8/93 1:23:45 PM); if the number has only an integer part, displays it as a date; if the number has only a fractional part, displays it as time
Long Date	Displays a Long Date; Long Date is defined in the International section of the Control Panel
Medium Date	Displays the date using the month abbreviation, without the day of the week (for example, 08-Nov-93)
Short Date	Displays a Short Date; Short Date is defined in the International section of the Control Panel
Long Time	Displays a Long Time; Long Time is defined in the International section of the Control Panel and includes hours, minutes, and seconds
Medium Time	Does not display seconds; displays hours in 12-hour format and uses the AM/PM designator
Short Time	Does not display seconds; uses 24-hour format and no AM/PM designator

In a user-defined format for a date, the *format* specification contains a series of tokens. Each token is replaced in the output string by its appropriate value.

A date can be output by using a combination of the following tokens:

Token	Output
c	The equivalent of the format *ddddd ttttt*. Read the definitions that follow.
ddddd	The date including the day, month, and year according to the machine's current Short Date setting; the default Short Date setting for the United States is *m/d/yy.*
dddddd	The date including the day, month, and year according to the machine's current Long Date setting; the default Long Date setting for the United States is *mmmm dd, yyyy.*
ttttt	The time including the hour, minute, and second using the machine's current time settings; the default time format is *h:mm:ss* AM/PM.

Finer control over the output is available by including *format* tokens that deal with the individual components of the date-time. These tokens are:

Token	Output
d	The day of the month as a one or two digit number (1–31)
dd	The day of the month as a two digit number (01–31)
ddd	The day of the week as a three letter abbreviation (Sun–Sat)
dddd	The day of the week without abbreviation (Sunday–Saturday)
w	The day of the week as a number (Sunday as 1, Saturday as 7)
ww	The week of the year as a number (1–53)
m	The month of the year or the minute of the hour as a one or two digit number. The minute is output if the preceding token was an hour; otherwise, the month is output.
mm	The month or the year or the minute of the hour as a two digit number. The minute is output if the preceding token was an hour; otherwise, the month is output.
mmm	The month of the year as a three letter abbreviation (Jan–Dec)
mmmm	The month of the year without abbreviation (January–December)

Token	Output
`q`	The quarter of the year as a number (1–4)
`y`	The day of the year as a number (1–366)
`yy`	The year as a two-digit number (00–99)
`yyyy`	The year as a three- or four-digit number (100–9999)
`h`	The hour as a one- or two-digit number (0–23)
`hh`	The hour as a two-digit number (00–23)
`n`	The minute as a one- or two-digit number (0–59)
`nn`	The minute as a two-digit number (00–59)
`s`	The second as a one- or two-digit number (0–59)
`ss`	The second as a two-digit number (00–59)

By default, times display using a military (24-hour) format. Several tokens are provided in date time format specifications to change this default. They use a 12-hour format. These are:

Token	Output
`AM/PM`	An uppercase AM with any hour before noon; an uppercase PM with any hour between noon and 11:59 PM
`am/pm`	A lowercase am with any hour before noon; a lowercase pm with any hour between noon and 11:59 PM
`A/P`	An uppercase A with any hour before noon; an uppercase P with any hour between noon and 11:59 PM
`a/p`	A lowercase a with any hour before noon; a lowercase p with any hour between noon and 11:59 PM
`AMPM`	The contents of the 1159 string (s1159) in the WIN.INI file with any hour before noon; the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. Note: `ampm` is equivalent to `AMPM.`

Any set of characters can be inserted into the output by enclosing them in double quotes. Any single character can be inserted by preceding it with a backslash, "\".

Formatting Strings

By default, string formatting transfers characters from left to right. The exclamation point, !, when added to the format specification causes characters to be transferred from right to left. By default, characters being transferred are not modified. The less than, <, and the greater than, >, characters force case conversion on the transferred characters. Less than forces output characters to be in lowercase. Greater than forces output characters to be in uppercase.

Character transfer is controlled by the at sign, @, and the ampersand, &, characters in the format specification. These operate as follows:

Character	Interpretation
@	Output a character or a space; if there is a character in the string being formatted in the position where the @ appears in the format string, display it; otherwise, display a space in that position.
&	Output a character or nothing; if there is a character in the string being formatted in the position where the & appears, display it; otherwise, display nothing.

A format specification for strings can have one or two sections separated by a semicolon. If you use one section, the format applies to all string data. If you use two sections, the first section applies to string data, the second to Null values and zero-length strings.

Examples This example demonstrates some of the string-formatting tokens.

```
Sub Button1_Click
   Dim msgtext As String
   msgtext = Format("Section #AB-234", "<\[&&&&&&&&&&&&&&&\]") _
   & Chr$(13) & Chr$(13) & Format("incoming", ">@@@@@@@@\!\!") _
   & Chr$(13) & Chr$(13) _
   & Format("Profits are expected to rise.", _
   "!&&&&&&&&&&&&&&&&&")
End Sub
```

This example calculates the square root of 2 as a double-precision floating point value and displays it in scientific notation.

```
Sub Button1_Click
   Dim value As Double
   Dim msgtext As String
   value = CDbl(Sqr(2))
   msgtext = "The square root of 2 is " & Format(value,
"Scientific")
End Sub
```

This example uses several different date-formatting tokens to format the result of the Now function, which returns the current date and time on the computer's clock.

```
Sub ClickMe_Click
   dim msgtext As String
   msgtext = Now & Chr$(13) & Chr$(13) _
   & "Today is " & Format(Now, "dddd") & ", " _
   & Format(Now, "mmmm") & " " & Format(Now, "dd") & ", " _
   & Format(Now, "yyyy") & "." _
   & Chr$(13) & "The time is " & Format(Now, "h:nn am/pm") _
   & " and " & Format(Now, "s") & " seconds."
End Sub
```

For other examples of the Format function, read "CCur Function" on page 115, "FV Function" on page 216, and "GoTo Statement" on page 253.

See Also

"Asc Function" on page 88
"CCur Function" on page 115
"CDbl Function" on page 117
"Chr Function" on page 121
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144
"Str Function" on page 462

FreeFile Function

This standard VB function returns the lowest unused file number.

Syntax FreeFile

Argument	Description
Not applicable	

Returns The lowest file number not in use.

Usage The FreeFile function is used when you need to supply a file number and want to make sure that you are not choosing a file number that is already in use.

The value returned can be used in a subsequent Open statement.

Example This example opens a file and assigns to it the next file number available.

```
Sub Button_Click
   Dim filenumber As Integer
   Dim filename As String
   filenumber = FreeFile
   filename = "d:\temp\trace.txt"
   On Error Resume Next
   Open filename For Input As filenumber
   If Err <> 0 then
      Exit Sub
   End If
   Close #filenumber
End Sub
```

See Also "Open Statement" on page 344

Function...End Function Statement

This standard VB construct defines a function procedure.

Syntax [Static] [Private] Function *name*([[Optional]*parameter* [As *type*]][, ...]) [As *funcType*]

name = *expression*

End Function

Placeholder	Description
name	The name of the function
parameter	The argument to pass to the function when it is called
type	The data type for the argument
funcType	The data type for the value returned by the function

Returns The value calculated by the *expression*; the program line *name* = *expression* assigns the return value to the name of the function.

Usage The purpose of a function is to produce and return a single value of a specified type. Recursion is supported.

The data type of *name* determines the type of the return value. Use a type character as part of the *name*, or use the As *funcType* clause to specify the data type. Otherwise the default data type is variant. When calling the function, you need not specify the type character.

The *parameters* are specified as a comma-separated list of variable names. The data type of a parameter can be specified by using a type character or by using the As clause. Record parameters are declared using an As clause and a *type* that has previously been defined using the Type statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the Function statement. Every reference to an array parameter within the body of the function must have a consistent number of dimensions.

You specify the return value for the function name using the *name* = *expression* assignment, where *name* is the name of the function and *expression* evaluates to a return value. If omitted, the value returned is 0 for numeric functions and an empty string ("") for string functions and vartype 0 (Empty) is returned for a return type of variant. The function returns to the caller when the End Function statement is reached or when an Exit Function statement is executed.

If you declare a parameter as Optional, a procedure can omit its value when calling the function. Only parameters with variant data types can be declared as optional, and optional arguments must appear after the required arguments in the Function statement. The function IsMissing must be used to check whether an optional parameter was omitted by the user or not. Named parameters are described under the Call statement heading, but they can be used when the function is used in an expression as well.

The Static keyword specifies that the variables declared within the function retain their values as long as the program is running, regardless of the way the variables are declared.

The Private keyword specifies that the function is not accessible to functions and subprograms from other modules. Only procedures defined in the same module have access to a Private function.

Basic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it modifies the variable passed by the caller. Use this feature with great care.

Use Sub to define a procedure with no return value.

CAUTION: You cannot write your own functions or subprograms directly in the methods and events exposed in Siebel Tools. You can write functions and subprograms in the (general) (declarations) section of a given method script. However, if you want your routines to be available throughout the program, you can use the Application_PreInvokeMethod or an external DLL file as a central place to write them. For details, read Siebel Technical Notes #207 and #217.

If you create more than one function or subprogram in the (general) (declarations) section, be sure that any function or subprogram that may be called by other user-defined functions and subprograms appears before the procedure that calls it. Otherwise, you cannot compile your procedures.

Example This example declares a function that is later called by the main subprogram. The function performs a calculation on the value sent to it, thereby changing the value of the variable. For other examples, read “Declare Statement” on page 155, and the second example within “GoTo Statement” on page 253.

```
(general) (declarations)
Option Explicit
Declare Function Calculate(i as Single) As Single

Function Calculate(i As Single)
   i = i * 3 + 2
   Calculate = i
End Function

Sub Button_Click
   Dim x as String
   Dim y As Single
   x = 34
   y = val(x)
   Call Calculate(y)
End Sub
```

See Also
“Call Statement” on page 112
“Dim Statement” on page 161
“Global Statement” on page 249
“IsMissing Function” on page 279
“Option Explicit Statement” on page 352
“Static Statement” on page 460
“Sub...End Sub Statement” on page 467

FV Function

This standard VB function returns the future value for a constant periodic stream of cash flows as in an annuity or a loan.

Syntax FV(*rate, nper, pmt, pv, due*)

Argument	Description
rate	The interest rate per period
nper	The total number of payment periods
pmt	The constant periodic payment per period
pv	The present value or the initial lump sum amount paid (as in the case of an annuity) or received (as in the case of a loan)
due	An integer value indicating when the payments are due (0 = end of each period, 1 = beginning of the period)

Returns A number representing the future value of an investment such as an annuity or loan.

Usage The given interest rate is assumed constant over the life of the annuity.

If payments are on a monthly schedule and the annual percentage rate on the annuity or loan is 9%, the *rate* is 0.0075 (.0075 = .09/12).

Example This example finds the future value of an annuity, based on terms specified by the user.

```
Sub Button_Click
   Dim aprate, periods
   Dim payment, annuitypv
   Dim due, futurevalue
   Dim msgtext
   annuitypv = 100000
   aprate = 6.75
   If aprate >1 then
      aprate = aprate/100
   End If
```

```
   periods = 60
   payment = 10000
   ' Assume payments are made at end of month
   due = 0
   futurevalue = FV(aprate/12,periods,-payment,-annuitypv,due)
   msgtext = "The future value is: " & Format(futurevalue,
"Currency")
End Sub
```

See Also

"IPmt Function" on page 271
"IRR Function" on page 273
"NPV Function" on page 333
"Pmt Function" on page 356
"PPmt Function" on page 361
"PV Function" on page 369
"Rate Function" on page 375

Get Statement

This standard VB function reads data from a file opened in Random or Binary mode and puts it in a variable.

Syntax Get [#]*filenumber,* [*recnumber*], *varName*

Argument	Description
filenumber	The file number used in the Open statement to open the file
recnumber	An expression of type long containing either the number of the record at which to start reading in Random mode, or the offset of the byte at which to start reading in Binary mode
varName	The name of the variable into which Get reads file data; *varName* can be any variable except Object or Array variables, although single array elements can be used

Returns Not applicable

Usage For more information about how files are numbered when they are opened, read "Open Statement" on page 344.

The *Recnumber* argument is in the range 1 to 2,147,483,647. If this argument is omitted, the next record or byte is read.

NOTE: The commas before and after the *recnumber* are required, even if you do not supply a *recnumber.*

For Random mode, the following rules apply:

Blocks of data are read from the file in chunks whose size is equal to the size specified in the Len clause of the Open statement. If the size of *varName* is smaller than the record length, the additional data is discarded. If the size of *varName* is larger than the record length, an error occurs.

For variable length string variables, Get reads two bytes of data that indicate the length of the string, then reads the data into *varName.*

For variant variables, Get reads two bytes of data that indicate the type of the variant, then it reads the body of the variant into *varName*. Note that variants containing strings contain two bytes of data type information followed by two bytes of length followed by the body of the string.

User defined types are read as if each member were read separately, except no padding occurs between elements.

Files opened in Binary mode behave similarly to those opened in Random mode, except that:

- Get reads variables from the disk without record padding.
- Variable-length Strings that are not part of user-defined types are not preceded by the two-byte string length. Instead, the number of bytes read is equal to the length of *varName*.

Example This example opens a file for Random access, gets its contents, and closes the file again. The second subprogram, `createfile`, creates the `c:\temp001` file used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   ' Put the numbers 1-10 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 1 to 10
      Write #1, x
   Next x
   Close #1
End Sub

Sub Button1_Click
   Dim acctno as String * 3
   Dim recno as Long
   Dim msgtext as String
   Call CreateFile
   recno = 1
   newline = Chr(10)
   Open "c:\temp001" For Random As #1 Len = 3
   msgtext = "The account numbers are:" & newline
```

```
      Do Until recno = 11
            Get #1,recno,acctno
            msgtext = msgtext & acctno
            recno = recno + 1
      Loop
      Close #1
         Kill "c:\temp001"
   End Sub
```

See Also

"Open Statement" on page 344
"Put Statement" on page 367
"Type Statement" on page 487

GetAssocBusComp Method

GetAssocBusComp returns the association business component. The association business component can be used to operate on the association using the normal business component mechanisms. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetAttr Function

This standard VB function returns the attributes of a file, folder, or volume label.

Syntax

GetAttr(*pathname*)

Argument	Description
pathname	A string or string expression evaluating to the name of the file, folder, or volume label to query

Returns

An integer representing a file attribute. The file attributes returned by GetAttr are as follows:

Value	Meaning
0	Normal file
1	Read-only file
2	Hidden file
4	System file
8	Volume label
16	Directory (folder)
32	Archive—file has changed since last backup

Usage

Pathname can contain drive and folder information, but cannot contain wildcards (* and ?).

If GetAttr returns a value other than those in the preceding list, the return value represents the sum of the return values for those attributes that are set; thus, for example, a return value of 6 represents a hidden system file.

See Also

"FileAttr Function" on page 190
"SetAttr Statement" on page 413

GetBusComp Method

The GetBusComp method returns the specified Siebel business component. It is used with Siebel business objects. For details, read *Siebel Object Interfaces Reference*.

GetBusObject Method

The GetBusObject method instantiates and returns a new instance of the argument specified business object. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

GetChild Method

GetChild returns a specified child property set of a property set. For details, read *Siebel Object Interfaces Reference*.

GetChildCount Method

GetChildCount returns the number of child property sets attached to a parent property set. For details, read *Siebel Object Interfaces Reference*.

GetFieldValue Method

GetFieldValue returns the value for the argument-specified field for the current record of a Siebel business component. Use this method to access a field value. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetFirstProperty Method

GetFirstProperty retrieves the name of the first property of a business service or property set. For details, read *Siebel Object Interfaces Reference*.

GetFormattedFieldValue Method

GetFormattedFieldValue returns the field value in the current local format; it returns values in the same format as the Siebel UI. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetMultipleFieldValues Method

GetMultipleFieldValues() allows users to retrieve the field values for a particular record as specified in the property set input argument. For details, read *Siebel Object Interfaces Reference*.

GetMVGBusComp Method

GetMVGBusComp returns the MVG business component associated with a Siebel business component field. This business component can be used to operate on the multi-value group using the normal business component mechanisms. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetNamedSearch Method

GetNamedSearch returns the named search specification specified by *searchName*. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetNextProperty Method

When the name of the first property has been retrieved, this method retrieves the name of the next property of a business service. For details, read *Siebel Object Interfaces Reference*.

GetObject Function

This standard VB function returns a COM object associated with the file name or the application name.

Syntax A GetObject(*pathname*)

Syntax B GetObject(*pathname, class*)

Syntax C GetObject(, *class*)

Argument	Description
pathname	The full path and filename for the object to retrieve
class	A string containing the class of the object

Returns The object associated with *pathname* or the object associated with *class*.

Usage Use GetObject with the Set statement to assign a variable to the object for use in a Basic procedure. The variable used must first be dimensioned as an object.

Syntax A of GetObject accesses a COM object stored in a file. For example, the following two lines dimension a variable as an object and assign the object `payables.xls` to it. `Payables.xls` is located in the `My Documents folder`:

```
Dim oFileObject As Object
Set oFileObject = GetObject("C:\My Documents\payables.xls")
```

If the application supports accessing component objects within the file, you can append an exclamation point and a component object name to the file name, as follows:

```
Dim oComponentObject As Object
Set oComponentObject = _
  GetObject("C:\My Documents\payables.xls!R1C1: R13C9")
```

Syntax B of GetObject accesses a COM object of a particular class that is stored in a file. *Class* uses the syntax *appName.objectType,* where *appName* is the name of the application that provides the object, and *objectType* is the type or class of the object. For example:

```
Dim oClassObject As Object
Set oClassObject = GetObject("C:\My _
  Documents\payables.xls", "Excel.Sheet")
```

The third form of GetObject accesses the active COM object of a particular class. For example:

```
Dim oApplication As _
  SiebelHTMLApplication
Set oApplication = _
  GetObject(,"SiebelHTML.SiebelHTMLApplication.1")
```

If you use the third form of GetObject with an empty string ("") as the *pathname,* a new object instance of the specified type is returned. Thus, the preceding example gets an open instance of the Siebel application, while

```
Set oApplication = _
  GetObject("","SiebelHTML.SiebelHTMLApplication.1")
```

instantiates the Siebel application in memory, independent of the user interface.

NOTE: The last two examples refer to the object SiebelAppServer, which has been defined as an object type as configured in your external Visual Basic environment.

Example This example opens a specific Excel worksheet and places the contents of the Name field of the active business component in it. The worksheet file must already exist.

```
Sub Button1_Click
   Dim ExcelSheet As Object
   Set ExcelSheet = GetObject("C:\demo\test.xls")

   'Make Excel visible through the Application object.
   ExcelSheet.Application.Visible = 1

   'Place some text in the first cell of the sheet.
   ExcelSheet.ActiveSheet.Cells(1, 1).Value = _
      theApplication.ActiveBusComp.GetFieldValue("Name")
```

```
    'Save the sheet.
    ExcelSheet.Save
    'Close Excel with the Quit method on the Application object.
      +ExcelSheet.Application.Quit
  End Sub
```

See Also

"CreateObject Function" on page 134
"Is Operator" on page 275
"Me" on page 311
"New Operator" on page 325
"Nothing Function" on page 329
"Object Class" on page 336
"Typeof Function" on page 489

GetPicklistBusComp Method

GetPicklistBusComp returns the pick business component associated with the specified field in the current business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetProfileAttr Method

GetProfileAttr returns the value of an attribute in a user profile. For details, read *Siebel Object Interfaces Reference*.

GetProperty Method

The GetProperty method returns the value of the property whose name is specified in its argument on the object on which it is invoked. For details, read *Siebel Object Interfaces Reference*.

GetPropertyCount Method

GetPropertyCount() returns the number of properties associated with a property set. For details, read *Siebel Object Interfaces Reference*.

GetSearchExpr Method

GetSearchExpr returns the current search expression for a Siebel business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetSearchSpec Method

GetSearchSpec returns the search specification for the field specified in its argument. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetService Method

The GetService method returns a specified business service. If the service is not already running, it is constructed. For details, read *Siebel Object Interfaces Reference*.

GetSharedGlobal Method

The GetSharedGlobal method gets the shared user-defined global variables. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

GetType Method

GetType retrieves the value stored in the type attribute of a property set. For details, read *Siebel Object Interfaces Reference*.

GetUserProperty Method

GetUserProperty returns the value of a named UserProperty. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

GetValue Method

The GetValue method returns the value of a control in a Siebel applet, or of the value attribute of a property set. If used with control objects, the type of the return value depends on the specific control object. For details, read *Siebel Object Interfaces Reference*.

GetViewMode Method

GetViewMode returns the current visibility mode for a Siebel business component. This affects which records are returned by queries according to the visibility rules. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Global Statement

This standard VB statement declares Global variables for use in a Basic program.

Syntax Global *variableName* [As *type*] [,*variableName* [As *type*]] ...

Argument	Description
variableName	A variable name
type	The variable's data type

Returns Not applicable

Usage In Siebel VB, a Global variable must generally be declared in every module from which you wish to access that variable. Declare Global variables in the (general) (declarations) section for the module.

Basic is a strongly typed language: variables must be given a data type or they are assigned a type of variant.

If the As clause is not used, the type of the global variable can be specified by using a type character as a suffix to *variableName*. The two different type-specification methods can be intermixed in a single Global statement (although not on the same variable).

Regardless of which mechanism you use to declare a global variable, you can choose to use or omit the type character when referring to the variable in the rest of your program. The type suffix is not considered part of the variable name.

The available data types are:

- Arrays
- Numbers
- Records
- Strings
- Variants

Arrays

The available data types for arrays are numbers, strings, variants, and records. Arrays of arrays, dialog box records, and objects are not supported.

Array variables are declared by including a subscript list as part of the *variableName*. The syntax to use for *variableName* is:

```
Global variable([ subscriptRange, ... ]) [As typeName]
```

where *subscriptRange* is of the format:

```
[startSubscript To] endSubscript
```

If *startSubscript* is not specified, 0 is used as the default. The Option Base statement can be used to change the default to 1.

Both the *startSubscript* and the *endSubscript* are valid subscripts for the array. The maximum number of subscripts that can be specified in an array definition is 60.

If no *subscriptRange* is specified for an array, the array is declared as a dynamic array. In this case, the ReDim statement must be used to specify the dimensions of the array before the array can be used.

Numbers

Numeric variables can be declared using the As clause and one of the following numeric types: currency, integer, long, single, and double. Numeric variables can also be declared by including a type character as a suffix to the name.

Records

Record variables are declared by using an As clause and a *type* that has previously been defined using the Type statement. The syntax to use is:

```
Global variableName As typeName
```

Records are made up of a collection of data elements called fields. These fields can be of any numeric, string, variant, or previously defined record type. For details on accessing fields within a record, read "Type Statement" on page 487.

You cannot use the Global statement to declare a dialog record.

Strings

Siebel VB supports two types of strings, fixed-length and dynamic. Fixed-length strings are declared with a specific length (between 1 and 32767) and cannot be changed later. Use the following syntax to declare a fixed-length string:

```
Global variableName As String * length
```

Dynamic strings have no declared length, and can vary in length from 0 to 32767. The initial length for a dynamic string is 0. Use the following syntax to declare a dynamic string:

```
Global variableName$
```

or

```
Global variableName As String
```

Variants

Declare variables as variants when the type of the variable is not known at the start of, or might change during, the procedure. For example, a variant is useful for holding input from a user when valid input can be either text or numbers. Use the following syntax to declare a variant:

```
Global variableName
```

or

```
GlobalvariableName As Variant
```

Variant variables are initialized to vartype Empty.

Example This example contains two subroutines that share the variables *total* and *acctno*, and the record *grecord*.

```
(general)(declarations)
Option Explicit
Type acctrecord
   acctno As Integer
End Type
```

```
Global acctno as Integer
Global total as Integer
Global grecord as acctrecord
Declare Sub CreateFile

Sub CreateFile
   Dim x
   x = 1
   grecord.acctno = 2345
   Open "c:\temp001" For Output as #1
   Do While grecord.acctno <> 0
      grecord.acctno = 0
      If grecord.acctno <> 0 then
         Print #1, grecord.acctno
         x = x + 1
      End If
   Loop
   total = x-1
   Close #1
End Sub

Sub Button_Click
   Dim msgtext
   Dim newline as String
   newline = Chr$(10)
   Call CreateFile
   Open "c:\temp001" For Input as #1
   msgtext = "The new account numbers are: " & newline
   For x = 1 to total
      Input #1, grecord.acctno
      msgtext = msgtext & newline & grecord.acctno
   Next x
   Close #1
      Kill "c:\temp001"
End Sub
```

See Also

"Const Statement" on page 131
"Dim Statement" on page 161
"Option Base Statement" on page 347
"ReDim Statement" on page 377
"Static Statement" on page 460
"Type Statement" on page 487

GoTo Statement

This standard VB method transfers program control to a specified label.

Syntax GoTo *label*

Argument	Description
label	A name beginning in the first column of a line of code and ending in a colon (:)

Returns Not applicable

Usage A *label* has the same format as any other Basic name. Reserved words are not valid labels.

GoTo cannot be used to transfer control out of the current Function or Subprogram.

Example This example displays the date for one week from the date entered by the user. If the date is invalid, the GoTo statement sends program execution back to the beginning.

```
Sub Button_Click
   Dim str1 as String
   Dim nextweek
   Dim msgtext
start:
   str1 = "5/20/2001"
   answer = IsDate(str1)
   If answer = -1 then
      str1 = CVDate(str1)
      nextweek = DateValue(str1) + 7
      msgtext = "One week from the date entered is "
      msgtext = msgtext & Format(nextweek,"dddddd")
   Else
```

```
      GoTo start
      End If
End Sub
```

NOTE: Good programming practice is to avoid the use of GoTo statements. When possible, other constructs should be used to accomplish the same end. For example, the previous example could be reworked so that the If statement appears in a separate function called by the main program. If the test failed, the initial routine could be called again. The following example demonstrates this alternative.

```
(general) (declarations)
Option Explicit
' Variables must be declared in this section so that they
' can be used by both procedures.
Dim str1 As String, nextweek, MsgText As String
Declare Function CheckResponse(Answer) As String

Function CheckResponse(Answer) As String
   str1 = CVDate(str1)
   nextweek = DateValue(str1) + 7
   CheckResponse = "One week from the date entered is " & _
      Format(nextweek, "dddddd")
End Function

Sub Button1_Click
   Dim Answer as String
   str1 = "2/5/2001"
   Answer = IsDate(str1)
   If Answer <> -1 Then
      'Invalid date or format. Try again.
      Button1_Click
   Else
      Answer = CheckResponse(Answer)
      End If
End Sub
```

See Also

"Do...Loop Statement" on page 169
"If...Then...Else Statement" on page 259
"Select Case Statement" on page 403
"While...Wend Statement" on page 507

GotoView Method

GotoView activates the named view and its business object. As a side effect, this method activates the view's primary applet and its business component and activates the primary applet's first tab sequence control. Further, this method deactivates any business object, business component, applet, or control objects that were active prior to this method call. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

Hex Function

This standard VB function returns the hexadecimal representation of a number, as a string.

Syntax Hex[$](*number*)

Argument	Description
number	Any numeric expression

Returns The hexadecimal representation of *number* as a string.

Usage If *number* is an integer, the return string contains up to four hexadecimal digits; otherwise, the value is converted to a long integer, and the string can contain up to 8 hexadecimal digits.

To represent a hexadecimal number directly, precede the hexadecimal value with `&H`. For example, `&H10` equals decimal 16 in hexadecimal notation.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example This example returns the hex value for a number entered by the user.

```
Sub Button_Click
   Dim usernum as Integer
   Dim hexvalue as String
   usernum = 23
   hexvalue = Hex(usernum)
End Sub
```

See Also “Format Function” on page 202
“Oct Function” on page 338

Hour Function

This standard VB function returns the hour-of-day component (0–23) of a date-time value.

Syntax Hour(*time*)

Argument	Description
time	Any numeric or string expression that can evaluate to a date-time or time value

Returns If the expression evaluates to a date-time or time value, the hour component of that value; otherwise 0.

Usage *Time* can be any type, including string, and the Hour function attempts to convert *time* to a date value.

The return value is a variant of vartype 2 (integer). If the value of *time* is Null, a variant of vartype 1 (null) is returned.

For Hour to function without an error, the values passed to it must be in some form that can be interpreted as a time or date-time value. Thus, 13:26, or 1:45:12 PM returns valid results, but 1326 returns a 0.

Time is a double-precision value. The numbers to the left of the decimal point denote the date and the decimal value denotes the time (from 0 to .99999). Use the TimeValue function to obtain the correct value for a specific time.

See Also

"Date Statement" on page 147
"DateSerial Function" on page 149
"DateValue Function" on page 151
"Day Function" on page 153
"Minute Function" on page 316
"Month Function" on page 320
"Now Function" on page 331
"Second Function" on page 397
"Time Statement" on page 475
"TimeSerial Function" on page 479
"TimeValue Function" on page 481
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

If...Then...Else Statement

This standard VB control structure executes alternative blocks of program code based on one or more expressions.

Syntax A If *condition* Then *then_statement* [Else *else_statement*]

Syntax B If *condition* Then

statement_block

[ElseIf *expression* Then

statement_block]...

[Else

statement_block]

End If

Placeholder	Description
condition	Any expression that evaluates to TRUE (non-zero) or FALSE (zero)
then_statement	Any valid single expression
else statement	Any valid single expression
expression	Any expression that evaluates to TRUE (non-zero) or FALSE (zero)
statement_block	0 or more valid expressions, separated by colons (:), or on different lines

Returns Not applicable

Usage When multiple statements are required in either the Then or Else clause, use the block version (Syntax B) of the If statement.

Example This example checks the time and the day of the week and returns an appropriate message.

```
Sub Button_Click
   Dim h, m, m2, w
   h = hour(now)
   If h > 18 then
      m = "Good evening, "
   Elseif h >12 then
      m = "Good afternoon, "
   Else
      m = "Good morning, "
   End If
      w = weekday(now)
   If w = 1 or w = 7
      Then m2 = "the office is closed."
      Else m2 = "please hold for company operator."
      End If
End Sub
```

See Also

"Do...Loop Statement" on page 169
"GoTo Statement" on page 253
"On...GoTo Statement" on page 340
"Select Case Statement" on page 403
"While...Wend Statement" on page 507

Input Function

This standard VB function returns a string containing the characters read from a file.

Syntax Input[$](*number*, [#]*filenumber*)

Argument	Description
number	An integer representing the number of characters (bytes) to read from the file
filenumber	The number identifying the open file to use

Returns The data read from the file, as a string.

Usage The file pointer is advanced the number of characters read. Unlike the Input statement, the Input function returns every character it reads, including carriage returns, line feeds, and leading spaces.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

The input buffer can hold a maximum of 32 K characters. Be aware of this limit when attempting to pass in large amounts of data.

See Also
"Get Statement" on page 218
"Input Statement" on page 262
"Line Input Statement" on page 297
"Open Statement" on page 344
"Write Statement" on page 512

Input Statement

This standard VB statement reads data from a sequential file and assigns the data to variables.

Syntax Input [#]*filenumber, variable*[*, variable*]...

Argument	Description
filenumber	The file number used in the Open statement to open the file from which to read
variable	One or more variables to contain the values read from the file

Returns Not applicable

Usage The *filenumber* is the number used in the Open statement to open the file. The list of *variables* is separated by commas.

Example This example prompts a user for an account number, opens a file, searches for the account number, and displays the matching letter for that number. It uses the Input statement to increase the value of x and at the same time get the letter associated with each value. The second subprogram, `CreateFile`, creates the file `c:\temp001` used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Global x as Integer
Global y(100) as String

Sub CreateFile
' Put the numbers 1-10 and letters A-J into a file
   Dim startletter
   Open "c:\temp001" for Output as #1
   startletter = 65
   For x = 1 to 10
      y(x) = Chr(startletter)
      startletter = startletter + 1
   Next x
```

```
   For x = 1 to 10
      Write #1, x,y(x)
   Next x
   Close #1
End Sub

Sub Button2_Click
   Dim acctno as Integer
   Dim msgtext
   Call CreateFile
start: acctno = 2
   If acctno<1 Or acctno>10 then
      Goto start:
   End if
   x = 1
   Open "c:\temp001" for Input as #1
   Do Until x = acctno
      Input #1, x,y(x)
   Loop
      msgtext = "The letter for account number " & x & " is: " _
         & y(x)
   Close #1
      Kill "C:\TEMP001"
End Sub
```

See Also

"Get Statement" on page 218
"Input Function" on page 261
"Line Input Statement" on page 297
"Open Statement" on page 344
"Write Statement" on page 512

InsertChildAt Method

InsertChildAt inserts a child property set into a parent property set at a specific location. For details, read *Siebel Object Interfaces Reference*.

InStr Function

This standard VB function returns the position of the first occurrence of one string within another string.

Syntax A InStr([*start*,] *string1*, *string2*)

Syntax B InStr(*start, string1*, *string2*[, *compare*])

Argument	Description
start	An integer representing the position in *string1* to begin the search, with the first character in the string as 1
string1	The string to search
string2	The string to find
compare	0 if a case-sensitive search is desired 1 if a case-insensitive search is desired

Returns The position of the first character of *string2* in *string1*.

Usage If not specified, the search starts at the beginning of the string (equivalent to a *start* of 1). These arguments can be of any type. They are converted to strings.

InStr returns a zero under the following conditions:

- *start* is greater than the length of *string2*.
- *string1* is a null string.
- *string2* is not found.

If either *string1* or *string2* is a null variant, Instr returns a null variant.

If *string2* is a null string (""), Instr returns the value of *start*.

If *compare* is 0, a case-sensitive comparison based on the ANSI character set sequence is performed. If *compare* is 1, a case-insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your computer. If *compare* is omitted, the module level default, as specified with Option Compare, is used.

Example This example generates a random string of characters, then uses InStr to find the position of a single character within that string.

```
Sub Button_Click
   Dim x as Integer
   Dim y
   Dim str1 as String
   Dim str2 as String
   Dim letter as String
   Dim randomvalue
   Dim upper, lower
   Dim position as Integer
   Dim msgtext, newline
   upper = Asc("z")
   lower = Asc("a")
   newline = Chr(10)
   For x = 1 to 26
      Randomize
      randomvalue = Int(((upper - (lower + 1)) * Rnd) + lower)
      letter = Chr(randomvalue)
      str1 = str1 & letter
'Need to waste time here for fast processors
      For y = 1 to 1000
      Next y
   Next x
   str2 = "i"
   position = InStr(str1,str2)
   If position then
      msgtext = "The position of " & str2 & " is: " _
         & position & newline & "in string: " & str1
   Else
      msgtext = "The letter: " & str2 & " was not found in: " _
         & newline
      msgtext = msgtext & str1
      End If
End Sub
```

See Also

"Left Function" on page 290
"Mid Function" on page 312
"Mid Statement" on page 314
"Option Compare Statement" on page 350
"Right Function" on page 388
"Str Function" on page 462
"StrComp Function" on page 463

Int Function

This standard VB function returns the integer part of a number.

Syntax Int(*number*)

Argument	Description
number	Any numeric expression

Returns The integer part of *number*.

Usage For positive numbers, Int removes the fractional part of the expression and returns the integer part only. For negative numbers, Int returns the largest integer less than or equal to the expression. For example, Int (6.2) returns 6; Int(-6.2) returns -7.

The return type matches the type of the numeric expression. This includes variant expressions that return a result of the same vartype as input, except vartype 8 (string) returns as vartype 5 (double) and vartype 0 (empty) returns as vartype 3 (long).

The effect of this function is the same as that of the Fix function, except in the handling of negative numbers. Thus:

- Fix(-8.347) = -8
- Int(-8.347) = -9

Example This example uses Int to generate random numbers in the range between the ASCII values for lowercase a and z (97 and 122). The values are converted to letters and displayed as a string.

```
Sub Button_Click
   Dim x As Integer, y As Integer
   Dim str1 As String, letter As String
   Dim randomvalue As Double
   Dim upper As Integer, lower As Integer
   Dim msgtext, newline
   upper = Asc("z")
   lower = Asc("a")
```

```
      newline = Chr(10)
      For x = 1 to 26
         Randomize
         randomvalue = Int(((upper - (lower + 1)) * Rnd) + lower)
         letter = Chr(randomvalue)
         str1 = str1 & letter
      'Need to waste time here for fast processors
         For y = 1 to 1500
         Next y
      Next x
      msgtext = "The string is:" & newline
         msgtext = msgtext & str1
   End Sub
```

See Also

"Exp Function" on page 188
"Fix Function" on page 197
"Log Function" on page 303
"Rnd Function" on page 392
"Sgn Function" on page 430
"Sqr Function" on page 459

InvokeMethod Method

The InvokeMethod method calls a specialized method on an object that is not part of the object's interface. It may be used with applet, business component, business object business service, web applet, and application objects. When used with a business service, it may be used to implement a user-defined method. For details, read *Siebel Object Interfaces Reference*.

IPmt Function

This standard VB function returns the interest portion of a payment for a given period of an annuity.

Syntax IPmt(*rate, period, nper, pv, fv, due*)

Argument	Description
rate	Interest rate per period
period	The specific payment period, in the range 1 through *nper*
nper	The total number of payment periods
pv	The present value of the initial lump sum paid (as in an annuity) or received (as in a loan)
fv	The future value of the final lump sum required (as in a savings plan) or paid (which is 0 in a loan)
due	0 if payments are due at the end of the payment period 1 if payments are due at the beginning of the payment period

Returns The interest portion of a payment for a given payment period.

Usage The given interest rate is assumed to be constant over the life of the annuity. If payments are on a monthly schedule, then *rate* is 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the interest portion of a loan payment amount for payments made in the last month of the first year. The loan is for $25,000 to be paid back over 5 years at 9.5% interest.

```
Sub Button_Click
   Dim aprate, periods
   Dim payperiod
   Dim loanpv, due
   Dim loanfv, intpaid
   Dim msgtext
   aprate = .095
   payperiod = 12
```

```
    periods = 120
    loanpv = 25000
    loanfv = 0
    ' Assume payments are made at end of month
    due = 0
    intpaid = IPmt(aprate/12,payperiod,periods, _
    loanpv,loanfv,due)
    msgtext = "For a loan of $25,000 @ 9.5% for 10 years," _
     & Chr(10)
    msgtext = msgtext + "the interest paid in month 12 is: "_
          & Format(intpaid, "Currency")
End Sub
```

See Also

"FV Function" on page 216
"IRR Function" on page 273
"NPV Function" on page 333
"Pmt Function" on page 356
"PPmt Function" on page 361
"PV Function" on page 369
"Rate Function" on page 375

IRR Function

This standard VB function returns the internal rate of return for a stream of periodic cash flows.

Syntax IRR(*valuearray( )*, *guess*)

Argument	Description
valuearray( )	An array containing cash-flow values
guess	An estimate of the value returned by IRR

Returns The internal rate of return for a stream of periodic cash flows.

Usage *Valuearray( )* must have at least one positive value (representing a receipt) and one negative value (representing a payment). Payments and receipts must be represented in the exact sequence. The value returned by IRR varies with the change in the sequence of cash flows.

In general, a *guess* value of between 0.1 (10 percent) and 0.15 (15 percent) is a reasonable estimate.

IRR is an iterative function. It improves a given guess over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

Example This example calculates an internal rate of return (expressed as an interest rate percentage) for a series of business transactions (income and costs). The first value entered must be a negative amount, or IRR generates an “Illegal Function Call” error.

```
Sub Button_Click
   Dim cashflows() as Double
   Dim guess, count as Integer
   Dim i as Integer
   Dim intnl as Single
   Dim msgtext as String
   guess = .15
   count = 2
```

```
   ReDim cashflows(count + 1)
   For i = 0 to count-1
      cashflows(i) = 3000
   Next i
   intnl = IRR(cashflows(),guess)
   msgtext = "The IRR for your cash flow amounts is: "
      msgtext = msgtext & Format(intnl, "Percent")
End Sub
```

See Also

"FV Function" on page 216
"IPmt Function" on page 271
"NPV Function" on page 333
"Pmt Function" on page 356
"PPmt Function" on page 361
"PV Function" on page 369
"Rate Function" on page 375

Is Operator

Compares two object expressions and returns -1 if they refer to the same object, 0 otherwise.

Syntax

objectExpression Is *objectExpression*

Argument	Description
objectExpression	Any valid object expression

Returns

Not applicable

Usage

Is can also be used to test if an object variable has been set to Nothing.

Example

For examples of the Is operator, read "CreateObject Function" on page 134 and "GetObject Function" on page 234.

See Also

"CreateObject Function" on page 134
"GetObject Function" on page 234
"Me" on page 311
"Nothing Function" on page 329
"Object Class" on page 336
"Typeof Function" on page 489

IsDate Function

This standard VB function indicates whether or not an expression is a legal date.

Syntax

IsDate(*expression*)

Argument	Description
expression	Any valid expression

Returns

-1 (TRUE) if *expression* is a legal date, 0 (FALSE) if it is not.

Usage

IsDate returns -1 (TRUE) if the expression is of vartype 7 (date) or a string that can be interpreted as a date.

Example

This example adds a number to today's date value and checks to see if it is still a valid date (within the range January 1, 100 AD, through December 31, 9999 AD).

```
Sub Button_Click
   Dim curdatevalue
   Dim yrs
   Dim msgtext
   curdatevalue = DateValue(Date$)
   yrs = 20
   yrs = yrs * 365
   curdatevalue = curdatevalue + yrs
   If IsDate(curdatevalue) = -1 then
      msgtext = Format(CVDate(curdatevalue))
   Else
      "The date is not valid."
   End If
End Sub
```

See Also

"CVDate Function" on page 144
"IsEmpty Function" on page 277
"IsNull Function" on page 281
"IsNumeric Function" on page 283
"VarType Function" on page 497

IsEmpty Function

This standard VB function is used to determine whether a variable of data type variant has been initialized.

Syntax IsEmpty(*expression*)

Argument	Description
expression	Any expression containing a variable of data type variant

Returns -1 (TRUE) if a variant has been initialized; 0 (FALSE) otherwise.

Usage IsEmpty returns -1 (TRUE) if the variant is of vartype 0 (empty). Any newly defined variant defaults to being of Empty type, to signify that it contains no initialized data. An Empty variant converts to zero when used in a numeric expression, or an empty string ("") in a string expression.

Example This example prompts for a series of test scores and uses IsEmpty to determine whether the maximum allowable limit has been hit. (IsEmpty determines when to exit the Do...Loop.)

```
Sub Button_Click
   Dim arrayvar(10)
   Dim x as Integer
   Dim tscore as Single
   Dim total as Integer
   x = 1
   Do
      tscore = 88
      arrayvar(x) = tscore
      x = x + 1
   Loop Until IsEmpty(arrayvar(10)) <> -1
   total = x-1
   msgtext = "You entered: " & Chr(10)
   For x = 1 to total
       msgtext = msgtext & Chr(10) & arrayvar(x)
      Next x
End Sub
```

See Also

"IsDate Function" on page 276
"IsNull Function" on page 281
"IsNumeric Function" on page 283
"VarType Function" on page 497

IsMissing Function

This standard VB function is used to determine whether an optional argument for a procedure has been supplied by the caller.

Syntax IsMissing(*argname*)

Argument	Description
argname	An optional argument for a subprogram, function, Siebel VB statement, or Siebel VB function

Returns -1 (TRUE) if an optional parameter was not supplied by the user; 0 (FALSE) otherwise.

Usage IsMissing is used in procedures that have optional arguments to find out whether the argument's value was supplied or not.

Example This example prints a list of uppercase characters. The quantity printed is determined by the user. If the user wants to print every character, the Function `myfunc` is called without any argument. The function uses IsMissing to determine whether to print every uppercase character or just the quantity specified by the user.

```
Function myfunc(Optional arg1)
   If IsMissing(arg1) = -1 then
      arg1 = 26
   End If
   msgtext = "The letters are: " & Chr$(10)
   For x = 1 to arg1
      msgtext = msgtext & Chr$(x + 64) & Chr$(10)
   Next x
End Function

Sub Button_Click
   Dim arg1
   arg1 = 0
   If arg1 = 0 then
      myfunc()
   Else
```

```
        myfunc(arg1)
        End If
End Sub
```

See Also "Function...End Function Statement" on page 213

IsNull Function

This standard VB function is used to determine whether a variant variable has the Null value.

Syntax

IsNull(*expression*)

Argument	Description
expression	Any expression containing a variable of data type variant

Returns

-1 (TRUE) if a variant expression contains the Null value, 0 (FALSE) otherwise.

Usage

Null variants have no associated data and serve only to represent invalid or ambiguous results. Null is not the same as Empty, which indicates that a variant has not yet been initialized.

Example

This example asks for ten test score values and calculates the average. If any score is negative, the value is set to Null. Then IsNull is used to reduce the total count of scores (originally 10) to just those with positive values before calculating the average.

```
Sub Button_Click
   Dim arrayvar(10)
   Dim count as Integer
   Dim total as Integer
   Dim x as Integer
   Dim tscore as Single
   count = 10
   total = 0
   For x = 1 to count
      tscore = 88
      If tscore<0 then
         arrayvar(x) = Null
      Else
         arrayvar(x) = tscore
         total = total + arrayvar(x)
      End If
   Next x
   Do While x <> 0
      x = x - 1
```

```
            If IsNull(arrayvar(x)) = -1 then
               count = count-1
            End If
         Loop
         msgtext = "The average (excluding negative values) is: "
            msgtext = msgtext & Chr(10) & Format(total/count, "##.##")
      End Sub
```

See Also

"IsDate Function" on page 276
"IsEmpty Function" on page 277
"IsNumeric Function" on page 283
"VarType Function" on page 497

IsNumeric Function

This standard VB function is used to determine whether the value of a variable is numeric.

Syntax IsNumeric(*expression*)

Argument	Description
expression	Any valid expression

Returns -1 (TRUE) if *expression* has a data type of Numeric, 0 (FALSE) otherwise.

Usage IsNumeric returns -1 (TRUE) if the expression is of vartypes 2-6 (numeric) or a string that can be interpreted as a number.

If numeric input is required, IsNumeric can be used to determine whether the value input by the user is a valid number before converting the input to a numeric data type for processing.

See Also "IsDate Function" on page 276
"IsEmpty Function" on page 277
"IsNull Function" on page 281
"VarType Function" on page 497

Kill Statement

Deletes files from a hard disk or floppy drive.

Syntax Kill *pathname*

Argument	Description
pathname	A string expression that represents a valid DOS file specification

Returns Not applicable

Usage The *pathname* specification can contain paths and wildcards (? and *). Kill deletes files only, not folders. To delete folders, use the RmDir function.

Example This example prompts a user for an account number, opens a file, searches for the account number, and displays the matching letter for that number. The second subprogram, `CreateFile`, creates the file `c:\temp001` used by the main subprogram. After processing is done, the first subroutine uses Kill to delete the file.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile
Global x as Integer
Global y(100) as String

Sub CreateFile
' Put the numbers 1-10 and letters A-J into a file
   Dim startletter
   Open "c:\temp001" for Output as #1
   startletter = 65
   For x = 1 to 10
      y(x) = Chr(startletter)
      startletter = startletter + 1
   Next x
   For x = 1 to 10
      Write #1, x,y(x)
   Next x
      Close #1
End Sub
```

```
Sub Button_Click
   Dim acctno as Integer
   Dim msgtext
   Call CreateFile
i: acctno = 6
   If acctno<1 Or acctno>10 then
      Goto i:
   End if
   x = 1
   Open "c:\temp001" for Input as #1
   Do Until x = acctno
      Input #1, x,y(x)
   Loop
      msgtext = "The letter for account number " & x & " is: _
      " & y(x)
   Close #1
      kill "c:\temp001"
End Sub
```

See Also

“FileAttr Function” on page 190
“FileDateTime Function” on page 194
“GetAttr Function” on page 222
“RmDir Statement” on page 390

LastRecord Method

LastRecord moves to the last record in a business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

LBound Function

This standard VB function returns the lower bound of the subscript range for an array.

Syntax LBound(*arrayname* [, *dimension*])

Argument	Description
arrayname	The name of the array to query
dimension	The dimension to query

Returns The lower bound (lowest index number) of dimension *dimension* of *arrayname*.

Usage The dimensions of an array are numbered starting with 1. If the *dimension* is not specified, 1 is the default.

LBound can be used with UBound to determine the length of an array.

Example This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1

Sub Button_Click
   Dim arrayvar() as Integer
   Dim count as Integer
   Dim answer as String
   Dim x, y as Integer
   Dim total
   total = 0
   x = 1
   count = 4
   ReDim arrayvar(count)
start:
   Do until x = count + 1
    arrayvar(x) = 98
    x = x + 1
   Loop
   x = LBound(arrayvar,1)
```

```
        count = UBound(arrayvar,1)
        For y = x to count
            total = total + arrayvar(y)
           Next y
     End Sub
```

See Also "Dim Statement" on page 161
"Global Statement" on page 249
"Option Base Statement" on page 347
"ReDim Statement" on page 377
"Static Statement" on page 460
"UBound Function" on page 490

LCase Function

This standard VB function returns a lowercase copy of a string.

Syntax LCase[$](*string*)

Argument	Description
string	A string or an expression containing a string

Returns A copy of *string*, with uppercase letters converted to lowercase.

Usage The substitution of characters is based on the country specified in the Windows Control Panel. LCase accepts expressions of type string. LCase accepts any type of argument and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

The LCase or UCase function is useful for comparing string data when you need to compare the actual text values, but the case in which input is entered is irrelevant.

Example This example converts a string entered by the user to lowercase.

```
Sub Button_Click
      Dim userstr as String
   userstr = "This Is A Test"
   userstr = LCase$(userstr)
End Sub
```

See Also "UCase Function" on page 492

Left Function

This standard VB function returns a string of a specified length copied from the beginning of another string.

Syntax Left[$](*string*, *length*)

Argument	Description
string	A string, or an expression containing a string, from which a portion is to be copied
length	An integer representing the number of characters to copy

Returns A substring of *string*, of length *length*, beginning at the first character of *string*.

Usage If *length* is greater than the length of *string*, Left returns the whole string.

Left accepts expressions of type string. Left accepts any type of *string*, including numeric values, and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

Example This example extracts a user's first name from the entire name entered.

```
Sub Button_Click
      Dim username as String
      Dim count as Integer
      Dim firstname as String
      Dim charspace
      charspace = Chr(32)
      username = "Chris Smith"
      count = InStr(username,charspace)
         firstname = Left(username,count)
End Sub
```

See Also

"Len Function" on page 292
"LTrim Function" on page 309
"Mid Function" on page 312
"Mid Statement" on page 314
"Right Function" on page 388
"RTrim Function" on page 396
"Str Function" on page 462
"StrComp Function" on page 463
"Trim Function" on page 486

Len Function

This standard VB function returns the length of a string or string variable.

Syntax A Len(*string*)

Syntax B Len(*varName*)

Argument	Description
string	A string or an expression that evaluates to a string
varName	A variable that contains a string

Returns The length of *string* or the string contained in the variable *varName*.

Usage If the argument is a string, the number of characters in the string is returned. If the argument is a variant variable, Len returns the number of bytes required to represent its value as a string; otherwise, the length of the built-in data type or user-defined type is returned.

If syntax B is used, and *varName* is a variant containing a NULL, Len returns a Null variant.

Example This example returns the length of a name entered by the user (including spaces).

```
Sub Button_Click
      Dim username as String
      username = "Chris Smith"
      count = Len(username)
End Sub
```

See Also "InStr Function" on page 265

Let (Assignment Statement)

The Let statement is a standard VB statement that assigns an expression to a Basic variable.

Syntax [Let] *variable* = *expression*

Placeholder	Description
variable	The variable to which a value is to be assigned
expression	The expression containing the value to be assigned to *variable*

Returns Not applicable

Usage The keyword Let is optional.

The Let statement can be used to assign a value or expression to a variable with a data type of numeric, string, variant, or record variable. You can also use the Let statement to assign to a record field or to an element of an array.

When assigning a value to a numeric or string variable, standard conversion rules apply.

Let differs from Set in that Set assigns a variable to a COM object. For example,

`Set o1 = o2` sets the object reference.

`Let o1 = o2` sets the value of the default member.

Example This example uses the Let statement for the variable sum. The subroutine finds an average of 10 golf scores.

```
Sub Button_Click
      Dim score As Integer
      Dim x, sum
      Dim msgtext
      Let sum = 34
      For x = 1 to 10
         score = 76
         sum = sum + score
```

```
        Next x
        msgtext = "Your average is: " & CInt(sum/(x-1))
End Sub
```

See Also

“Const Statement” on page 131
“Lset Statement” on page 307
“Service_InvokeMethod Event” on page 409

Like Operator

Like is a standard VB operator used to compare the contents of string expressions.

Syntax *string* LIKE *pattern*

Placeholder	Description
string	Any string or string expression
pattern	Any string expression to compare to *string*

Returns -1 (TRUE) if *string* matches *pattern*, 0 (FALSE) otherwise.

Usage *pattern* can include the following special characters:

Character	Matches
?	A single character
*	A set of zero or more characters
#	A single digit character (0–9)
[*chars*]	A single character in *chars*
[!*chars*]	A single character not in *chars*
[*startchar*–*endchar*]	A single character in the range *startchar* to *endchar*
[!*startchar*–*endchar*]	A single character not in the range *startchar* to *endchar*

Both ranges and lists can appear within a single set of square brackets. Ranges are matched according to their ANSI values. In a range, *startchar* must be less than *endchar*.

If either *string* or *pattern* is NULL, then the result value is NULL.

The Like operator respects the current setting of Option Compare.

For more information about operators, read "Expressions" on page 63.

Example This example tests whether a letter is lowercase.

```
Sub Button_Click
   Dim userstr as String
   Dim revalue as Integer
   Dim msgtext as String
   Dim pattern
   pattern = "[a-z]"
   userstr = "E"
   retvalue = userstr LIKE pattern
   If retvalue = -1 then
      msgtext = "The letter " & userstr & " is lowercase."
   Else
      msgtext = "Not a lowercase letter."
   End If
End Sub
```

See Also "InStr Function" on page 265
"Option Compare Statement" on page 350
"StrComp Function" on page 463

Line Input Statement

This standard VB statement reads a line from a sequential file into a string variable.

Syntax A Line Input [#] *filenumber*, *varName*

Syntax B Line Input [*prompt*,] *varName*

Argument	Description
filenumber	The file number, given in the Open statement, of the open file from which to read
varName	A string variable into which a line of data or user input is to be read
prompt	A string literal prompting for keyboard input

Returns Not applicable

Usage If it is included, the *filenumber* is the number used in the Open statement to open the file. If *filenumber* is not provided, the line is read from the keyboard.

If *prompt* is not provided, a question mark (?) is displayed as the prompt.

Line Input is used to read lines of text from a text file in which the data elements are separated by carriage returns. To read data from a file of comma-separated values, use Read.

Example This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, `CreateFile`, creates the file `C:\temp001` used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   Rem Put the numbers 1-10 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 1 to 10
```

```
      Write #1, x
   Next x
   Close #1
End Sub

Sub Button_Click
   Dim testscore as String
   Dim x
   Dim y
   Dim newline
   Call CreateFile
   Open "c:\temp001" for Input as #1
   x = 1
   newline = Chr(10)
   msgtext = "The contents of c:\temp001 is: " & newline
   Do Until x = Lof(1)
      Line Input #1, testscore
      x = x + 1
      y = Seek(1)
      If y>Lof(1) then
         x = Lof(1)
      Else
         Seek 1,y
      End If
      msgtext = msgtext & testscore & newline
   Loop
   Close #1
      Kill "c:\temp001"
End Sub
```

See Also

"Get Statement" on page 218
"Input Function" on page 261
"Input Statement" on page 262
"Open Statement" on page 344

Loc Function

This standard VB function returns the current offset within an open file.

Syntax Loc(*filenumber*)

Argument	Description
filenumber	The number given in the Open statement, of the open file to query

Returns For random files, the number of the last record read or written; for files opened in append, input, or output mode, the current byte offset divided by 128; for files opened in binary mode, the offset of the last byte read or written.

Example This example creates a file of account numbers as entered by the user. When the user finishes, the example displays the offset in the file of the last entry made.

```
Sub Button_Click
   Dim filepos as Integer
   Dim acctno() as Integer
   Dim x as Integer
   x = 0
   Open "c:\TEMP001" for Random as #1
   Do
      x = x + 1
      Redim Preserve acctno(x)
      acctno(x) = 234
      If acctno(x) = 0 then
         Exit Do
      End If
      Put #1,, acctno(x)
   Loop
   filepos = Loc(1)
   Close #1
      Kill "C:\TEMP001"
End Sub
```

See Also "Eof Function" on page 173
"Lof Function" on page 302
"Open Statement" on page 344

Lock Statement

This standard VB statement controls access to an open file.

Syntax Lock [#]*filenumber*[, [*start*] [To *end*]]

Argument	Description
filenumber	The file number of the open file as used in the Open statement
start	A long integer representing the number of the first record or byte offset to lock or unlock
end	A long integer representing the number of the last record or byte offset to lock or unlock

Returns Not applicable

Usage For binary mode, *start* and *end* are byte offsets. For random mode, *start* and *end* are record numbers. If *start* is specified without *end*, then only the record or byte at *start* is locked. If *end* is specified without *start*, then records or bytes from record number or offset 1 to *end* are locked.

For Input, output, and append modes, *start* and *end* are ignored and the whole file is locked.

Lock and Unlock always occur in pairs with identical parameters. Locks on open files must be removed before closing the file, or unpredictable results may occur.

Example This example locks a file that is shared by others on a network, if the file is already in use. The second subprogram, `CreateFile`, creates the file used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   ' Put the letters A-J into the file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
```

```
      For x = 1 to 10
         Write #1, Chr(x + 64)
      Next x
      Close #1
   End Sub

   Sub Button_Click
      Dim btngrp, icongrp
      Dim defgrp
      Dim answer
      Dim noaccess as Integer
      Dim msgabort
      Dim msgstop as Integer
      Dim acctname as String
      noaccess = 70
      msgstop = 16
      Call CreateFile
      On Error Resume Next
      btngrp = 1
      icongrp = 64
      defgrp = 0
      answer = 1
      If answer = 1 then
         Open "c:\temp001" for Input as #1
         If Err = noaccess then
               'File Locked -Aborted
         Else
            Lock #1
            Line Input #1, acctname
            Unlock #1
         End If
         Close #1
      End If
         Kill "C:\TEMP001"
   End Sub
```

See Also

"Open Statement" on page 344
"Unlock Statement" on page 494

Lof Function

This standard VB function returns the length in bytes of an open file.

Syntax Lof(*filenumber*)

Argument	Description
filenumber	The number of the open file, as used in the Open statement

Returns The length of the open file, in bytes.

Usage The *filenumber* is the number used in the Open statement that opened the file.

Example This example opens a file and prints its contents to the screen.

```
Sub Button_Click
   Dim fname As String,fchar() As String
   Dim x As Integer, msgtext As String, newline As String
   newline = Chr(10)
   fname = "d:\temp\trace.txt"
   On Error Resume Next
   Open fname for Input as #1
   If Err <> 0 then
      Exit Sub
   End If
   msgtext = "The contents of " & fname & " is: " _
      & newline & newline
   Redim fchar(Lof(1))
    For x = 1 to Lof(1)
       fchar(x) = Input(1,#1)
       msgtext = msgtext & fchar(x)
   Next x
      Close #1
End Sub
```

See Also
"Eof Function" on page 173
"FileLen Function" on page 195
"Loc Function" on page 299
"Open Statement" on page 344

Log Function

This standard VB function returns the natural logarithm of a number.

Syntax Log(*number*)

Argument	Description
number	Any valid numeric expression

Returns The natural logarithm of *number.*

Usage The return value is single-precision for an integer, currency, or single-precision numeric expression; double precision for a long, variant, or double-precision numeric expression.

Example This example uses the Log function to determine which number is larger: 999^1000 (999 to the 1000th power) or 1000^999 (1000 to the 999th power). Note that you cannot use the exponent (^) operator for numbers this large.

```
Sub Button_Click
   Dim x
   Dim y
   x = 999
   y = 1000
   a = y * (Log(x))
   b = x * (Log(y))
   If a>b then
       "999^1000 is greater than 1000^999"
   Else
       "1000^999 is greater than 999^1000"
      End If
End Sub
```

See Also “Exp Function” on page 188
“Fix Function” on page 197
“Int Function” on page 268
“Rnd Function” on page 392
“Sgn Function” on page 430
“Sqr Function” on page 459

LoginId Method

The LoginId method returns the login id of the user who started the Siebel applications. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

LoginName Method

The LoginName method returns the login name of the user who started the Siebel application (the name typed into the login dialog box). It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

LookupMessage Method

The LookupMessage method returns the translated string for the specified key, in the current language, from the specified category. For details, read *Siebel Object Interfaces Reference*.

Lset Statement

This standard VB statement copies one string to another or assigns a user-defined type variable to another.

Syntax A Lset *string* = *string-expression*

Syntax B Lset *variable1* = *variable2*

Argument	Description
string	A string variable or string expression to contain the copied characters
string-expression	A string variable or string expression containing the string to be copied
variable1	A variable within a user-defined type to contain the copied variable
variable2	A variable containing a user-defined type to be copied

Returns Not applicable

Usage If *string* is shorter than *string-expression*, Lset copies the leftmost characters of *string-expression* into *string*. The number of characters copied is equal to the length of *string*.

If *string* is longer than *string-expression,* every character in *string-expression* is copied into *string*, filling it from left to right. Leftover characters in *string* are replaced with spaces.

In Syntax B, the number of characters copied is equal to the length of the shorter of *variable1* and *variable2*.

Lset cannot be used to assign variables of different user-defined types if either contains a variant or a variable-length string.

Example This example puts a user's last name into the variable `*c`. If the name is longer than the size of `lastname`, then the user's name is truncated.

```
Sub Button_Click
   Dim lastname as String
   Dim strlast as String * 8
   lastname = "Smith"
   Lset strlast = lastname
   msgtext = "Your last name is: " & strlast
End Sub
```

See Also "Rset Statement" on page 394

LTrim Function

This standard VB function returns a string with leading spaces removed.

Syntax LTrim[$](*string*)

Argument	Description
string	A string or string expression containing the string to be trimmed

Returns A copy of *string* with leading space characters removed.

Usage LTrim accepts any type of *string*, including numeric values, and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function typically returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

Example This example trims the leading spaces from a string padded with spaces on the left.

```
Sub Button_Click
   Dim userinput as String
   Dim numsize
   Dim str1 as String * 50
   Dim strsize
   strsize = 50
   userinput = "abdcGFTRes"
   numsize = Len(userinput)
   str1 = Space(strsize-numsize) & userinput
   ' Str1 has a variable number of leading spaces.
   str1 = LTrim$(str1)
      ' Str1 now has no leading spaces.
End Sub
```

See Also

“Left Function” on page 290
“Mid Function” on page 312
“Mid Statement” on page 314
“Right Function” on page 388
“RTrim Function” on page 396
“Trim Function” on page 486

Me

Me is standard VB shorthand used to refer to the currently used object.

Syntax A

With Me

.methodname() statement

End With

Syntax B

Me.*methodname() statement*

Placeholder	Description
methodname	The name of the method to be used with the object
statement	The code to be executed, or the arguments to the method

Returns

Not applicable

Usage

Some Siebel VB modules are attached to application objects, and Siebel VB subroutines are invoked when such an application object encounters events. For example, Me may refer to a button that triggers a Basic routine when the user clicks on it, or when a method is invoked on an application object by a program statement.

Subroutines in such contexts can use the variable Me to refer to the object that triggered the event (for example, the button that was clicked). The programmer can use Me in the same way as any other object variable, except that Me cannot be Set.

Example

For examples, read "Service_InvokeMethod Event" on page 409 and "With Statement" on page 510.

See Also

"CreateObject Function" on page 134
"GetObject Function" on page 234
"New Operator" on page 325
"Nothing Function" on page 329
"Object Class" on page 336
"Typeof Function" on page 489

Mid Function

This standard VB function returns a portion of a string, starting at a specified location within the string.

Syntax Mid[$](*string*, *start*[, *length*])

Argument	Description
string	A string or string expression containing the string to be copied
start	An integer representing the starting position in *string* to begin copying characters
length	An integer representing the number of characters to copy

Returns A substring of *string*, of length *length,* beginning at the *start* character of *string.*

Usage Mid accepts any type of string, including numeric values, and converts the input value to a string. If the *length* argument is omitted or if *string* is smaller than *length*, then Mid returns computer characters in *string*. If *start* is larger than *string*, then Mid returns an empty string ("").

The index of the first character in a string is 1.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function typically returns a variant of vartype 8 (string). If the value of *string* is Null, a variant of vartype 1 (Null) is returned. Mid$ requires the string argument to be of type string or variant. Mid allows the string argument to be of any data type.

To modify a portion of a string value, read "Mid Statement" on page 314.

Example This example uses the Mid function to find the last name in a string entered by the user.

```
Sub Button_Click
   Dim username as String
   Dim position as Integer
   username = "Chris Smith"
   Do
```

```
        position = InStr(username," ")
        If position = 0 then
           Exit Do
        End If
        position = position + 1
        username = Mid(username,position)
        Loop
End Sub
```

See Also

"Left Function" on page 290
"Len Function" on page 292
"LTrim Function" on page 309
"Mid Function" on page 312
"Right Function" on page 388
"RTrim Function" on page 396
"Trim Function" on page 486

Mid Statement

Mid replaces part (or all) of one string with another, starting at a specified location.

Syntax Mid (*stringVar*, *start*[, *length*]) = *string*

Argument	Description
stringVar	The string to be changed
start	An integer representing the position in *stringVar* at which to begin replacing characters
length	An integer representing the number of characters to replace
string	The string to place into *stringVar*

Returns The value of *stringVar* with *string* embedded at the specified location.

Usage If the *length* argument is omitted, or if there are fewer characters in *string* than specified in *length*, then Mid replaces the characters from the *start* to the end of the *string*. If *start* is larger than the number of characters in the indicated *stringVar,* then Mid appends *string* to *stringVar.*

If *length* is greater than the length of *string*, then *length* is set to the length of *string.* If *start* is greater than the number of characters in *stringVar,* an illegal function call error occurs at runtime. If *length* plus *start* is greater than the length of *stringVar,* then only the characters up to the end of *stringVar* are replaced.

Mid never changes the number of characters in *stringVar.*

The index of the first character in a string is 1.

Example This example uses the Mid statement to replace the last name in a user-entered string with asterisks (*).

```
Sub Button_Click
   Dim username as String
   Dim position as Integer
   Dim count as Integer
   Dim uname as String
```

```
      Dim replacement as String
      username = "Chris Smith"
      uname = username
      replacement = "*"
      Do
         position = InStr(username," ")
         If position = 0 then
            Exit Do
         End If
         username = Mid(username,position + 1)
         count = count + position
      Loop
      For x = 1 to Len(username)
         count = count + 1
         Mid(uname,count) = replacement
         Next x
   End Sub
```

See Also

"LCase Function" on page 289
"Left Function" on page 290
"Len Function" on page 292
"LTrim Function" on page 309
"Mid Statement" on page 314
"Right Function" on page 388
"RTrim Function" on page 396
"Trim Function" on page 486

Minute Function

This standard VB function returns the minute component (0–59) of a date-time value.

Syntax Minute(*time*)

Argument	Description
time	Any numeric or string expression that can evaluate to a date-time or time value

Returns If the expression evaluates to a date-time or time value, the minute component of that value; otherwise 0.

Usage *Time* can be of any type, including strings, and the Minute function attempts to convert the input value to a date-time value.

For Minute to function without an error, the values passed to it must be in some form that can be interpreted as a time or date-time value. Thus, 13:26, or 1:45:12 PM returns valid results, but 1326 returns a 0.

The return value is a variant of vartype 2 (integer). If the value of *time* is null, a variant of vartype 1 (Null) is returned.

Example This example extracts just the time (hour, minute, and second) from a file's last modification date and time.

```
Sub Button_Click
   Dim filename as String
   Dim ftime
   Dim hr, min
   Dim sec
   Dim msgtext as String
      i: msgtext = "Enter a filename:"
   filename = "d:\temp\trace.txt"
   If filename = "" then
      Exit Sub
   End If
   On Error Resume Next
   ftime = FileDateTime(filename)
```

```
    If Err <> 0 then
       Goto i:
    End If
    hr = Hour(ftime)
    min = Minute(ftime)
    sec = Second(ftime)
End Sub
```

See Also

"Date Statement" on page 147
"DateSerial Function" on page 149
"DateValue Function" on page 151
"Day Function" on page 153
"Hour Function" on page 257
"Month Function" on page 320
"Now Function" on page 331
"Second Function" on page 397
"Time Statement" on page 475
"TimeSerial Function" on page 479
"TimeValue Function" on page 481
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

MkDir Statement

This standard VB statement creates a new folder.

Syntax MkDir [*drive*:][*folder*\\]*folder*

Argument	Description
drive:	(Optional) The name of the drive on which the folder is to be created, as a letter, or a string expression representing the drive name
folder\\	If the folder is not to be created on the current folder of the specified drive (or the default drive if none is specified), the path to the folder in which the new folder is to be created, or a string expression representing the path
folder	The name of the folder to be created, or a string expression representing the folder name

Returns Not applicable

Usage The *drive:* argument is optional. If *drive:* is omitted, MkDir creates the new folder on the current drive. If the *drive:* argument is used, it must include the colon.

Example This example makes a new temporary folder in C:\ and then deletes it.

```
Sub Button_Click
      Dim path as String
      On Error Resume Next
      path = CurDir(C)
      If path <> "C:\" then
         ChDir "C:\"
      End If
      MkDir "C:\TEMP01"
      If Err = 75 then
      Else
         RmDir "C:\TEMP01"
         End If
End Sub
```

See Also

"ChDir Statement" on page 118
"ChDrive Statement" on page 120
"CurDir Function" on page 141
"Dir Function" on page 167
"RmDir Statement" on page 390

Month Function

This standard VB function returns an integer for the month component (1–12) of a date-time value.

Syntax Month(*date*)

Argument	Description
date	Any numeric or string expression that can evaluate to a date-time or date value

Returns If the expression evaluates to a date-time or date value, the month component of that value; otherwise 0.

Usage *Date* can be of any type, including string, and the Month function attempts to convert the input value to a date-time value.

For Month to function without an error, the values passed to it must be in some form that can be interpreted as a time or date-time value. Thus, 11/20, or 11-20-2001 returns valid results, but 1120 returns a 0.

The return value is a variant of vartype 2 (integer). If the value of *date* is null, a variant of vartype 1 (null) is returned.

Example This example finds the month (1–12) and day (1–31) values for this Thursday.

```
Sub Button_Click
      Dim x As Integer, Today As Variant
      Dim msgtext
      Today = DateValue(Now)
      Let x = 0
      Do While Weekday(Today + x) <> 5
         x = x + 1
      Loop
      msgtext = "This Thursday is: " & Month(Today + x) &"/" _
            & Day(Today + x)
End Sub
```

See Also

"Date Statement" on page 147
"DateSerial Function" on page 149
"DateValue Function" on page 151
"Day Function" on page 153
"Hour Function" on page 257
"Minute Function" on page 316
"Now Function" on page 331
"Second Function" on page 397
"Time Statement" on page 475
"TimeSerial Function" on page 479
"TimeValue Function" on page 481
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

Name Method

The Name method returns the name of the object with which it is used. It can be used with applet, business component, business object, control, and application objects. For details, read *Siebel Object Interfaces Reference*.

Name Statement

This standard VB statement renames a file or copies a file from one folder to another.

Syntax Name [*path1*\]*oldfilename* As [*path2*\]*newfilename*

Argument	Description
path1\	A string expression containing the path to the current location of the file (must be entered if the file is not in the current folder of the current drive)
oldfilename	A string expression containing the name of the file to be renamed
path2\	A string expression containing the path to the location where the renamed file should appear; if a path is not given, the file goes in the current folder of the current drive
newfilename	A string expression containing the new name for the file

Returns Not applicable

Usage To be renamed, the file must be closed. If the file *oldfilename* is open or if the file *newfilename* already exists, Siebel VB generates an error message.

If this statement is used within the Siebel application, and no *path2*\ is specified, a copy of the original file goes in the `c:\siebel\bin` folder under the new name.

Example This example creates a temporary file, `c:\temp001`, renames the file to `c:\temp002`, then deletes them both. It calls the subprogram `CreateFile` to create the `c:\temp001` file.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
      Rem Put the numbers 1-10 into a file
      Dim x as Integer
      Dim y()
      Dim startletter
```

```
        Open "C:\TEMP001" for Output as #1
        For x = 1 to 10
           Write #1, x
        Next x
        Close #1
  End Sub

  Sub Button_Click
        Call CreateFile
        On Error Resume Next
        Name "C:\TEMP001" As "C:\TEMP002"
        Kill "TEMP001"
           Kill "TEMP002"
  End Sub
```

See Also

“FileAttr Function” on page 190
“FileCopy Statement” on page 192
“GetAttr Function” on page 222
“Kill Statement” on page 284

New Operator

The New operator allocates and initializes a new COM object of the named class.

Syntax

Set *objectVar* = New *className*

Dim *objectVar* As New *className*

Argument	Description
objectVar	The COM object to allocate and initialize
className	The class to assign to the object

Returns

Not applicable

Usage

In the Dim statement, New marks *objectVar* so that a new object is allocated and initialized when *objectVar* is first used. If *objectVar* is not referenced, then no new object is allocated.

NOTE: An object variable that was declared with New allocates a second object if objectVar is Set to Nothing and referenced again.

See Also

"CreateObject Function" on page 134
"Dim Statement" on page 161
"Global Statement" on page 249
"Service_InvokeMethod Event" on page 409
"Static Statement" on page 460

NewPropertySet Method

The NewPropertySet method constructs a new property set object. For details, read *Siebel Object Interfaces Reference*.

NewRecord Method

NewRecord adds a new record (row) to a Siebel business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

NextRecord Method

NextRecord moves the current record to the next record in a Siebel business component, invoking any associated Basic events. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Nothing Function

This standard VB function removes an instantiated object from memory.

Syntax

Set *objectName* = Nothing

Argument	Description
objectName	The name of the object variable to set to Nothing

Returns

Not applicable

Usage

Nothing is the value object variables have when they do not refer to an object, either because they have not been initialized yet or because they were explicitly set to Nothing. For example:

```
If Not objectVar Is Nothing then
   objectVar.Close
Set objectVar = Nothing
End If
```

Use the Nothing function to remove an object that you have instantiated from memory.

Example

This example adds an activity record indicating that a contact has been added when a new contact is added in the Siebel application. It presumes that Contact is the parent business component and instantiates the Action business component, destroying it using the Nothing statement after the job is done. For other examples of the Nothing function, read "CreateObject Function" on page 134 and "GetObject Function" on page 234.

```
Sub BusComp_WriteRecord

   Dim oBCact as BusComp
   Set oBCact =
theApplication.ActiveBusObject.GetBusComp("Action")

   With oBCact
      .NewRecord NewAfter
      .SetFieldValue "Type", "Event"
      .SetFieldValue "Description", "ADDED THRU SVB"
```

```
            .SetFieldValue "Done", Format(Now(),"mm/dd/yyyy hh:mm:ss")
            .SetFieldValue "Status", "Done"
            .WriteRecord
         End With

            set oBCact = Nothing
      End Sub
```

See Also

"Is Operator" on page 275
"New Operator" on page 325

Now Function

This standard VB function returns the current date and time.

Syntax Now()

Argument	Description
Not applicable	

Returns The current date and time as indicated by the clock of the server that is executing the code.

Usage The Now function returns a variant of vartype 7 (date) that represents the current date and time according to the setting of the computer's date and time. Use the Format function to specify the format in which the date and time should appear.

Example This example finds the month (1–12) and day (1–31) values for this Thursday. For another example, read "Format Function" on page 202.

```
Sub Button_Click
   Dim x As Integer, today As Variant
   Dim msgtext As String
   Today = DateValue(Now)
   Let x = 0
   Do While Weekday(Today + x) <> 5
      x = x + 1
   Loop
   msgtext = "This Thursday is: " & Month(Today + x) & "/" & _
         Day(Today + x)
End Sub
```

See Also

"Date Function" on page 146
"Date Statement" on page 147
"Day Function" on page 153
"Hour Function" on page 257
"Minute Function" on page 316
"Second Function" on page 397
"Time Function" on page 473
"Time Statement" on page 475
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

NPV Function

This standard VB function returns the net present value of an investment based on a stream of periodic cash flows and a constant interest rate.

Syntax NPV(*rate, valuearray( )*)

Argument	Description
rate	The discount rate per period
valuearray( )	An array containing cash-flow values

Returns The net present value of cash flows in *valuarray( )* based on the *rate.*

Usage *Valuearray( )* must have at least one positive value (representing a receipt) and one negative value (representing a payment). Payments and receipts must be represented in the exact sequence. The value returned by NPV varies with the change in the sequence of cash flows.

If the discount rate is 12% per period, *rate* is the decimal equivalent, that is, 0.12.

NPV uses future cash flows as the basis for the net present value calculation. If the first cash flow occurs at the beginning of the first period, its value should be added to the result returned by NPV and must not be included in *valuearray()*.

Example This example finds the net present value of an investment, given a range of cash flows by the user.

See Also "FV Function" on page 216
"IPmt Function" on page 271
"IRR Function" on page 273
"Pmt Function" on page 356
"PPmt Function" on page 361
"PV Function" on page 369
"Rate Function" on page 375

Null Function

This standard VB function sets a variant variable to the Null value.

Syntax Null

Argument	Description
Not applicable	

Returns A variant value set to NULL.

Usage Null is used to set a variant variable to the Null value explicitly, as follows:

```
variableName = Null
```

Note that variants are initialized by Basic to the empty value, which is different from the Null value.

Example This example asks for ten test score values and calculates the average. If any score is negative, the value is set to Null. Then IsNull is used to reduce the total count of scores (originally 10) to just those with positive values before calculating the average.

```
Sub Button_Click
   Dim arrayvar(10)
   Dim count as Integer
   Dim total as Integer
   Dim x as Integer
   Dim tscore as Single
   count = 10
   total = 0
   For x = 1 to count
      tscore = 88
      If tscore < 0 then
         arrayvar(x) = Null
      Else
         arrayvar(x) = tscore
         total = total + arrayvar(x)
      End If
   Next x
```

```
      Do While x <> 0
         x = x - 1
         If IsNull(arrayvar(x)) = -1 then
            count = count - 1
         End If
      Loop
      msgtext = " The average (excluding negative values) is: "
         msgtext = msgtext & Chr(10) & Format (total/count, "##.##")
   End Sub
```

See Also

"IsEmpty Function" on page 277
"IsNull Function" on page 281
"VarType Function" on page 497

Object Class

Object is a class that provides access to COM automation objects.

Syntax Dim *variableName* As Object

Placeholder	Description
variableName	The name of the object variable to declare

Returns Not applicable

Usage To create a new object, first dimension a variable, using the Dim statement, then set the variable to the return value of CreateObject or GetObject, as follows:

```
Dim COM As Object

Set COM = CreateObject("spoly.cpoly")
```

To refer to a method or property of the newly created object, use the syntax *objectvar.property* or *objectvar.method*, as follows:

```
COM.reset
```

Example This example uses the special Siebel object class BusComp to declare the variables used for accessing the Account Contacts view within the Siebel application.

```
Sub Button1_Click
   Dim i as integer
   Dim icount as integer
   Dim oBC as BusComp

   ' BusObject returns the business object associated with a
   ' control or applet.
   ' GetBusComp returns a reference to a Siebel
   ' business component that is in the UI context

   set oBC = me.BusObject.GetBusComp("Contact")

   i = oBC.FirstRecord ' returns 0 if fails, 1 if succeeds
   if i <> 1 then
      TheRaiseErrorText "Error accessing contact records for the
```

```
account."
   else
      icount = 0
      ' NextRecord returns 1 if it successfully
      ' moved to the next record in the BC
      While i = 1
         icount = icount + 1
         i = oBC.NextRecord ' returns 1 if successful
      wend
      oBC.FirstRecord
      end if
End Sub
```

See Also

“CreateObject Function” on page 134
“GetObject Function” on page 234
“New Operator” on page 325
“Nothing Function” on page 329
“Typeof Function” on page 489

Oct Function

This standard VB function converts a number to an octal (base 8) number.

Syntax Oct[$](*number*)

Argument	Description
number	Any numeric expression

Returns The octal representation of a number, as a string.

Usage If the numeric expression has a data type of integer, the string contains up to six octal digits; otherwise, the expression is converted to a data type of long, and the string can contain up to 11 octal digits.

The dollar sign ($) in the function name is optional. If it is included, the return data type is string. Otherwise the function returns a variant of vartype 8 (string).

NOTE: To represent an octal number directly, precede the octal value with `&O`. For example, `&O10` equals decimal 8 in octal notation.

Example This example prints the octal values for the numbers from 1 to 15.

```
Sub Button_Click
   Dim x As Integer, y As Integer
   Dim msgtext As String
   Dim nofspaces As Integer
   msgtext = "Octal numbers from 1 to 15:" & Chr(10)
   For x = 1 to 15
      nofspaces = 10
      y = Oct(x)
      If Len(x) = 2 then
         nofspaces = nofspaces - 2
      End If
      msgtext = msgtext & Chr(10) & x & Space(nofspaces) & y
      Next x
End Sub
```

See Also "Hex Function" on page 256

On...GoTo Statement

This standard VB programming control structure causes execution to branch to a label in the current procedure based on the value of a numeric expression.

Syntax

On *number* GoTo *label1*[, *label2, ...*]

Argument	Description
number	Any numeric expression that evaluates to a positive number
label1, label2, ...	A label in the current procedure to branch to if *number* evaluates to 1, 2, and so on

Returns

Not applicable

Usage

If *number* evaluates to 0 or to a number greater than the number of labels following GoTo, the program continues at the next statement. If *number* evaluates to a number less than 0 or greater than 255, an “Illegal function call” error is issued.

See Also

“GoTo Statement” on page 253
“Select Case Statement” on page 403

On Error Statement

This standard VB statement specifies the location of an error-handling routine within the current procedure.

Syntax On Error {GoTo *label* | Resume Next | GoTo 0}

Returns Not applicable

Usage On Error is used to provide routines to handle specific errors. On Error can also be used to disable an error-handling routine. Unless an On Error statement is used, any run-time error is fatal; that is, Siebel VB terminates the execution of the program.

An On Error statement includes *one* of the following parts:

Part	Definition
GoTo *label*	Enables the error-handling routine that starts at *label*. If the designated label is not in the same procedure as the On Error statement, Siebel VB generates an error message.
Resume Next	Designates that error-handling code is handled by the statement that immediately follows the statement that caused an error. At this point, use the Err function to retrieve the error code of the run-time error.
GoTo 0	Disables any error handler that has been enabled.

When it is referenced by an On Error GoTo *label* statement, an error handler is enabled. When this enabling occurs, a run-time error results in program control switching to the error-handling routine and “activating” the error handler. The error handler remains active from the time the run-time error has been trapped until a Resume statement is executed in the error handler.

If another error occurs while the error handler is active, Siebel VB searches for an error handler in the procedure that called the current procedure (if this fails, Siebel VB looks for a handler belonging to the caller's caller, and so on). If a handler is found, the current procedure terminates, and the error handler in the calling procedure is activated.

NOTE: Because Siebel VB searches in the caller for an error handler, any additional On Error statements in the original error handler are ignored.

Executing an End Sub or End Function statement while an error handler is active is an error (No Resume). The Exit Sub or Exit Function statement can be used to end the error condition and exit the current procedure.

Example This example prompts the user for a drive and folder name and uses On Error to trap invalid entries.

```
Sub Button_Click
   Dim userdrive, userdir, msgtext
in1:
   userdrive = "c:"
   On Error Resume Next
   ChDrive userdrive
   If Err = 68 then
      Goto in1
   End If
in2:
   On Error Goto Errhdlr1
   userdir = "temp"
   ChDir userdrive & userdir
   userdir
   Exit Sub
Errhdlr1:
   Select Case Err
      Case 75
         msgtext = "Path is invalid."
      Case 76
         msgtext = "Path not found."
      Case 70
         msgtext = "Permission denied."
      Case Else
         msgtext = "Error " & Err & ": " & Error$ & "occurred."
```

```
        End Select
            Resume in2
    End Sub
```

See Also

"Erl Function" on page 177
"Err Function" on page 179
"Err Statement" on page 180
"Error Function" on page 182
"Error Statement" on page 184
"Resume Statement" on page 387

Open Statement

This standard VB statement opens a file for input or output.

Syntax Open *filename* [For *mode*] [Access *access*] [*lock*] As [#]*filenumber* [Len = *reclen*]

Argument	Description
filename	A string or string expression representing the name of the file to open
mode	A keyword indicating the purpose for which the file is opened
access	A keyword indicating the method of access to the file
lock	A keyword designating the access method allowed to the file by other processes
filenumber	An integer used to identify the file while it is open
reclen	In a Random or Binary file, the length of the records

Returns A file opened in the specified manner.

NOTE: The file opens in the default code page of the local operating system. File I/O does not support Unicode.

Usage The following keywords are used for mode, access, and lock:

Keyword	Consequences
Mode **Keywords**	
Input	Reads data from the file sequentially
Output	Puts data into the file sequentially
Append	Adds data to the file sequentially
Random	Gets data from the file by random access
Binary	Gets binary data from the file

Keyword	Consequences
Access **Keywords**	
Read	Reads data from the file only
Write	Writes data to the file only
Read Write	Reads or writes data to the file
Lock **Keywords**	
Shared	Read or write is available on the file
Lock Read	Only read is available
Lock Write	Only write is available
Lock Read Write	No read or write is available

A file must be opened before any input/output operation can be performed on it.

If *filename* does not exist, it is created when opened in append, binary, output, or random modes.

If *mode* is not specified, it defaults to random.

If *access* is not specified for random or binary modes, *access* is attempted in the following order: Read Write, Write, Read.

If *lock* is not specified, *filename* can be opened by other processes that do not specify a *lock*, although that process cannot perform any file operations on the file while the original process still has the file open.

Use the FreeFile function to find the next available value for *filenumber*.

The *reclen* parameter is ignored for Input, Output, and Append modes.

Example

This example opens a file for random access, gets the contents of the file, and closes the file again. The second subprogram, `CreateFile`, creates the file `c:\temp001` used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
      ' Put the numbers 1-10 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 1 to 10
      Write #1, x
   Next x
   Close #1
End Sub

Sub Button_Click
   Dim acctno as String * 3
   Dim recno as Long
   Dim msgtext as String
   Call CreateFile
   recno = 1
   newline = Chr(10)
   Open "c:\temp001" For Random As #1 Len = 3
   msgtext = "The account numbers are:" & newline
   Do Until recno = 11
         Get #1,recno,acctno
         msgtext = msgtext & acctno
         recno = recno + 1
   Loop
   Close #1
      Kill "c:\temp001"
End Sub
```

See Also "Close Statement" on page 129
"FreeFile Function" on page 212

Option Base Statement

This standard VB statement specifies the default lower bound to use for array subscripts.

Syntax Option Base *lowerBound*

Placeholder	Description
lowerBound	Either 0 or 1 or an expression that evaluates to one of these values

Returns Not applicable

Usage If no Option Base statement is specified, the default lower bound for array subscripts is 0.

The Option Base statement is *not* allowed inside a procedure and must precede any use of arrays in the module. Only one Option Base statement is allowed per module. It must be placed in the (general) (declarations) section in the Siebel VB Editor, as shown in Figure 1.

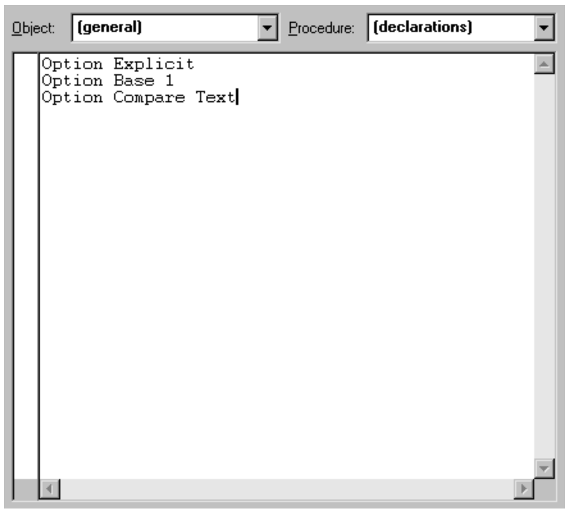


Figure 1. Placing Declarations in the (general) (declarations) Section

Example

This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1
Sub Button_Click
   Dim arrayvar() as Integer
   Dim count as Integer
   Dim answer as String
   Dim x, y as Integer
   Dim total
   total = 0
   x = 1
   count = 2
   ReDim arrayvar(count)
start:
   Do until x = count + 1
      arrayvar(x) = 87
```

```
            x = x + 1
        Loop
        x = LBound(arrayvar,1)
        count = UBound(arrayvar,1)
        For y = x to count
                total = total + arrayvar(y)
            Next y
    End Sub
```

See Also

"Dim Statement" on page 161
"Global Statement" on page 249
"LBound Function" on page 287
"ReDim Statement" on page 377
"Static Statement" on page 460

Option Compare Statement

This standard VB statement specifies the default method for string comparisons: either case-sensitive or case-insensitive.

Syntax Option Compare { Binary | Text }

Argument	Description
Not applicable	

Returns Not applicable

Usage The Option Compare statement must be placed in the (general) (declarations) section in the Siebel VB Editor, as shown in Figure 1 on page 348.

Binary comparisons are case-sensitive (that is, lowercase and uppercase letters are different). Text comparisons are not case-sensitive.

Binary comparisons compare strings based upon the ANSI character set. Text comparisons are based upon the relative order of characters as determined by the country code setting for your computer.

Example This example compares two strings: `JANE SMITH` and `jane smith`. When Option Compare is Text, the strings are considered the same. If Option Compare is Binary, they are not the same. Binary is the default. To see the difference, run the example, and then comment out the Option Compare statement and run it again.

```
Option Compare Text
Sub Button_Click
   Dim strg1 as String
   Dim strg2 as String
   Dim retvalue as Integer
   strg1 = "JANE SMITH"
   strg2 = "jane smith"
i:
   retvalue = StrComp(strg1,strg2)
   If retvalue = 0 then
      'The strings are identical
   Else
```

```
        'The strings are not identical
        Exit Sub
        End If
End Sub
```

See Also "InStr Function" on page 265
"StrComp Function" on page 463

Option Explicit Statement

This standard VB statement specifies that every variable in a module *must* be explicitly declared.

Syntax Option Explicit

Argument	Description
Not applicable	

Returns Not applicable

Usage By default, Basic declares any variables that do not appear in a Dim, Global, ReDim, or Static statement. Option Explicit causes such variables to produce a "Variable Not Declared" error.

Using the Option Explicit statement makes debugging code easier, because it forces you to declare variables before use. Good programming practice is to declare variables at the beginning of the unit within which they have scope (that is, at the beginning of the project, module, or procedure). Declaring variables in this manner simplifies finding their definitions when reading through code.

The Option Explicit statement must be placed in the (general) (declarations) section in the Siebel VB Editor, as shown in Figure 1 on page 348.

Example This example specifies that variables must be explicitly declared, thus preventing any mistyped variable names.

```
Option Explicit
Sub Button_Click
      Dim counter As Integer
      Dim fixedstring As String * 25
      Dim varstring As String
      '...(code here)...
End Sub
```

See Also

"Const Statement" on page 131
"Deftype Statement" on page 158
"Dim Statement" on page 161
"Function...End Function Statement" on page 213
"Global Statement" on page 249
"ReDim Statement" on page 377
"Static Statement" on page 460
"Sub...End Sub Statement" on page 467

ParentBusComp Method

ParentBusComp returns the parent (master) Siebel business component given the child (detail) business component of a Link. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Pick Method

The Pick method picks the currently selected record in a picklist business component (read "GetPicklistBusComp Method" on page 237) into the appropriate Fields of the parent business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Pmt Function

This standard VB function returns a constant periodic payment amount for an annuity or a loan.

Syntax Pmt(*rate, nper, pv, fv, due*)

Argument	Description
rate	The interest rate per period
nper	The total number of payment periods
pv	The present value of the initial lump sum amount paid (as with an annuity) or received (as with a loan)
fv	The future value of the final lump sum amount required (as with a savings plan) or paid (0 as with a loan)
due	0 if due at the end of each period 1 if due at the beginning of each period

Returns The constant periodic payment amount.

Usage *Rate* is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* is 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the monthly payment on a given loan.

```
Sub Button_Click
   Dim aprate, totalpay
   Dim loanpv, loanfv
   Dim due, monthlypay
   Dim yearlypay, msgtext
   loanpv = 25000
   aprate = 7.25
   If aprate >1 then
      aprate = aprate/100
   End If
   totalpay = 60
   loanfv = 0
```

```
'Assume payments are made at end of month
   due = 0
   monthlypay = Pmt(aprate/12,totalpay,-loanpv,loanfv,due)
   msgtext = "The monthly payment is: "
         Format(monthlypay, "Currency")
End Sub
```

See Also

"FV Function" on page 216
"IPmt Function" on page 271
"IRR Function" on page 273
"NPV Function" on page 333
"PPmt Function" on page 361
"PV Function" on page 369
"Rate Function" on page 375

PositionId Method

The PositionId property returns the position ID (ROW_ID from S_POSTN) of the user's current position. This is set by default when the Siebel application is started and may be changed (using Edit: Change Position) if the user belongs to more than one position. This method is used with the application object. For details, read *Siebel Object Interfaces Reference*.

PositionName Method

The PositionName property returns the position name of the user's current position. This is set by default when the Siebel application is started and may be changed (using Edit: Change Position) if the user belongs to more than one position. This method is used with the application object. For details, read *Siebel Object Interfaces Reference*.

PostChanges Method

PostChanges posts changes that are made in an applet. This method is used with applet objects. For details, read *Siebel Object Interfaces Reference*.

PPmt Function

This standard VB function returns the principal portion of the payment for a given period of an annuity.

Syntax PPmt(*rate, per, nper, pv, fv, due*)

Argument	Description
rate	The interest rate per period
per	The payment period, in the range from 1 to *nper*
nper	The total number of payment periods
pv	The present value of the initial lump sum amount paid (as with an annuity) or received (as with a loan)
fv	The future value of the final lump sum amount required (as with a savings plan) or paid (0 as with a loan)
due	0 if due at the end of each period 1 if due at the beginning of each period

Returns The principal portion of the payment for a given period.

Usage *Rate* is assumed to be constant over the life of the loan or annuity. If payments are on a monthly schedule, then *rate* is 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the principal portion of a loan payment amount for payments made in the last month of the first year. The loan is for $25,000 to be paid back over 5 years at 9.5% interest.

```
Sub Button_Click
   Dim aprate, periods
   Dim payperiod
   Dim loanpv, due
   Dim loanfv, principal
   Dim msgtext
   aprate = 9.5/100
   payperiod = 12
```

```
    periods = 120
    loanpv = 25000
    loanfv = 0
       ' Assume payments are made at end of month
    due = 0
    principal = PPmt(aprate/12,payperiod,periods, _
    -loanpv,loanfv,due)
    msgtext = "Given a loan of $25,000 @ 9.5% for 10 years,"
       msgtext = msgtext & Chr(10) & "the principal paid in month
12 is: "
End Sub
```

See Also

"FV Function" on page 216
"IPmt Function" on page 271
"IRR Function" on page 273
"NPV Function" on page 333
"PPmt Function" on page 361
"PV Function" on page 369
"Rate Function" on page 375

PreviousRecord Method

PreviousRecord moves to the previous record in a Siebel business component, invoking any associated Basic events. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Print Statement

This standard VB method prints data to an open file.

Syntax Print [#][*filenumber*,] *expressionList* [{;|, }]

Argument	Description
filenumber	The number of the open file to print to
expressionList	A list of values to be printed, in the form of literals or expressions

Returns Not applicable

Usage The Print statement outputs data to the specified *filenumber*. *Filenumber* is the number assigned to the file when it was opened. For more information, read "Open Statement" on page 344.

If the *expressionList* is omitted, a blank line is written to the file.

The values in *expressionList* may be separated by either a semicolon or a comma. A semicolon indicates that the next value should appear immediately after the preceding one without intervening white space. A comma indicates that the next value should be positioned at the next print zone. Print zones begin every 14 spaces.

The optional [{;|, }] argument at the end of the Print statement determines where output for the next Print statement to the same output file should begin. A semicolon places output immediately after the output from this Print statement on the current line; a comma starts output at the next print zone on the current line. If neither separator is specified, a CR-LF pair is generated and the next Print statement prints to the next line.

The Spc and Tab functions can be used inside a Print statement to insert a given number of spaces and to move the print position to a desired column, respectively.

The Print statement supports only elementary Basic data types. For more information on parsing this statement, read "Input Function" on page 261.

See Also

"Open Statement" on page 344
"Spc Function" on page 435
"Tab Function" on page 469
"Write Statement" on page 512

PropertyExists Method

PropertyExists returns a Boolean value indicating whether a specified property exists in a property set. For details, read *Siebel Object Interfaces Reference*.

Put Statement

This standard VB statement writes a variable to a file opened in random or binary mode.

Syntax Put [#]*filenumber,* [*recnumber*], *varName*

Argument	Description
filenumber	The file number used in the Open statement to open the file
recnumber	An expression of type long containing the record number or the byte offset at which to start writing
varName	The name of the variable containing the data to write

Returns Not applicable

Usage *Filenumber* is the number assigned to the file when it was opened. For more information, read "Open Statement" on page 344.

Recnumber is in the range 1 to 2,147,483,647. If *recnumber* is omitted, the next record or byte is written.

NOTE: The commas before and after *recnumber* are *required*, even if no *recnumber* is specified.

VarName can be any variable type except object, application data type, or array variables (single array elements can be used).

For random mode, the following conditions apply:

- Blocks of data are written to the file in chunks whose size is equal to the size specified in the Len clause of the Open statement. If the size of *varName* is smaller than the record length, the record is padded to the correct record size. If the size of the variable is larger than the record length, an error occurs.
- For variable length string variables, Put writes two bytes of data that indicate the length of the string and then writes the string data.

- For variant variables, Put writes two bytes of data that indicate the type of the variant; then it writes the body of the variant into the variable. Note that variants containing strings contain two bytes of type information, followed by two bytes of length, followed by the body of the string.
- User-defined types are written as if each member were written separately, except no padding occurs between elements.

Files opened in binary mode behave similarly to those opened in random mode except:

- Put writes variables to the disk without record padding.
- Variable-length Strings that are not part of user-defined types are not preceded by the two-byte string length.

NOTE: The Put statement uses the default code page of the local operating system. It does not write to the file in Unicode format.

Example

This example opens a file for Random access, puts the values 1 through 10 in it, prints the contents, and closes the file again.

```
Sub Button_Click
' Put the numbers 1-10 into a file
   Dim x As Integer, y As Integer
   Open "C:\TEMP001" as #1
   For x = 1 to 10
      Put #1,x, x
   Next x
   msgtext = "The contents of the file is:" & Chr(10)
   For x = 1 to 10
      Get #1,x, y
      msgtext = msgtext & y & Chr(10)
   Next x
   Close #1
      Kill "C:\TEMP001"
End Sub
```

See Also

"Close Statement" on page 129
"Get Statement" on page 218
"Open Statement" on page 344
"Write Statement" on page 512

PV Function

This standard VB function returns the present value of a constant periodic stream of cash flows as in an annuity or a loan.

Syntax PV(*rate, nper, pmt, fv, due*)

Argument	Description
rate	The interest rate per period
nper	The total number of payment periods
pmt	The constant periodic payment per period
fv	The future value of the final lump sum amount required (as with a savings plan) or paid (0 as with a loan)
due	0 if due at the end of each period 1 if due at the beginning of each period

Returns The present value of a constant periodic stream of cash flows.

Usage *Rate* is assumed constant over the life of the annuity. If payments are on a monthly schedule, then *rate* is 0.0075 if the annual percentage rate on the annuity or loan is 9%.

Example This example finds the present value of a 10-year $25,000 annuity that pays $1,000 a year at 9.5%.

```
Sub Button_Click
   Dim aprate As Integer, periods As Integer
   Dim payment As Double, annuityfv As Double
   Dim due As Integer, presentvalue As Double
   Dim msgtext
   aprate = 9.5
   periods = 120
   payment = 1000
   annuityfv = 25000
      ' Assume payments are made at end of month
   due = 0
   presentvalue = PV(aprate/12,periods,-payment, annuityfv,due)
```

```
   msgtext = "The present value for a 10-year $25,000 annuity @ 9.5%"
   msgtext = msgtext & " with a periodic payment of $1,000 is: "
      msgtext = msgtext & Format(presentvalue, "Currency")
End Sub
```

See Also

"FV Function" on page 216
"IPmt Function" on page 271
"IRR Function" on page 273
"NPV Function" on page 333
"Pmt Function" on page 356
"PPmt Function" on page 361
"Rate Function" on page 375

RaiseError Method

The RaiseError method raises a scripting error message to the browser. The error code is a canonical number. For details, read *Siebel Object Interfaces Reference*.

RaiseErrorText Method

The RaiseErrorText method raises a scripting error message to the browser. The error text is the specified literal string. For details, read *Siebel Object Interfaces Reference*.

Randomize Statement

This standard VB statement seeds the random number generator.

Syntax Randomize [*number*]

Argument	Description
number	An integer value between -32768 and 32767

Returns Not applicable

Usage If no *number* argument is given, Siebel VB uses the Timer function to initialize the random number generator.

Example This example generates a random string of characters using the Randomize statement and Rnd function. The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub Button_Click
   Dim x As Integer, y As Integer
   Dim str1 As String, str2 As String
   Dim letter As String
   Dim randomvalue
   Dim upper, lower
   Dim msgtext
   upper = Asc("z")
   lower = Asc("a")
   newline = Chr(10)
   For x = 1 to 26
      Randomize
      randomvalue = Int(((upper - (lower + 1)) * Rnd) + lower)
      letter = Chr(randomvalue)
      str1 = str1 & letter
      For y = 1 to 1500
      Next y
   Next x
      msgtext = str1
End Sub
```

See Also

"Rnd Function" on page 392
"Timer Function" on page 477

Rate Function

This standard VB function returns the interest rate per period for an annuity or a loan.

Syntax Rate(*nper, pmt, pv, fv, due, guess*)

Argument	Description
nper	The total number of payment periods
pmt	The constant periodic payment per period
pv	The present value of the initial lump sum amount paid (as with an annuity) or received (as with a loan)
fv	The future value of the final lump sum amount required (as with a savings plan) or paid (0 as with a loan)
due	0 if due at the end of each period 1 if due at the beginning of each period
guess	An estimate for the rate returned

Returns The interest rate per period.

Usage In general, a guess of between 0.1 (10 percent) and 0.15 (15 percent) would be a reasonable value for *guess*.

Rate is an iterative function: It improves the given value of *guess* over several iterations until the result is within 0.00001 percent. If it does not converge to a result within 20 iterations, it signals failure.

Example This example finds the interest rate on a 10-year $25,000 annuity that pays $100 per month.

```
Sub Button_Click
   Dim aprate
   Dim periods
   Dim payment, annuitypv
   Dim annuityfv, due
   Dim guess
```

```
      Dim msgtext as String
      periods = 120
      payment = 100
      annuitypv = 0
      annuityfv = 25000
      guess = .1
      ' Assume payments are made at end of month
      due = 0
      aprate = Rate(periods,-payment,annuitypv,annuityfv, _
      due, guess)
      aprate = (aprate * 12)
      msgtext = "The percentage rate for a 10-year $25,000 _
      annuity"
      msgtext = msgtext & "that pays $100/month has "
      msgtext = msgtext & "a rate of: " & Format(aprate, _
         "Percent")
   End Sub
```

See Also

“FV Function” on page 216
“IPmt Function” on page 271
“IRR Function” on page 273
“NPV Function” on page 333
“Pmt Function” on page 356
“PPmt Function” on page 361
“PV Function” on page 369

ReDim Statement

This standard VB statement changes the upper and lower bounds of a dynamic array's dimensions.

Syntax ReDim [Preserve] *arrayName* (*lower* To *upper*) [As [New] *type*], …

Argument	Description
arrayName	The name of the array to redimension
lower	The new lower bound for the array
upper	The new upper bound for the array
type	The data type for the array elements

Returns Not applicable

Usage ReDim reallocates memory for the dynamic array to support the specified dimensions, and can optionally re-initialize the array elements. ReDim cannot be used at the module level; it must be used inside of a procedure.

The Preserve option is used to change the last dimension in the array while maintaining its contents. If Preserve is not specified, the contents of the array are reinitialized. Numbers are set to zero (0). Strings and variants are set to empty ("").

If *lower* is not specified, 0 is used as the default. The Option Base statement can be used to change the default.

A dynamic array is normally created by using `Dim` to declare an array without a specified size. The maximum number of dimensions for a dynamic array created in this fashion is 8. If you need more than 8 dimensions, you can use the ReDim statement inside of a procedure to declare an array that has not previously been declared using Dim or Global. In this case, the maximum number of dimensions allowed is 60.

The available data types for arrays are numbers, strings, variants, records, and objects. Arrays of arrays, dialog box records, and objects are not supported.

If the As clause is not used, the type of the variable can be specified by using a type character as a suffix to the name. The two different type-specification methods can be intermixed in a single ReDim statement (although not on the same variable).

The ReDim statement cannot be used to change the number of dimensions of a dynamic array when the array has been given dimensions. It can change only the upper and lower bounds of the dimensions of the array. The LBound and UBound functions can be used to query the current bounds of an array variable's dimensions.

Care should be taken to avoid redimensioning an array in a procedure that has received a reference to an element in the array in an argument; the result is unpredictable.

Example

This example finds the net present value for a series of cash flows. The array variable that holds the cash flow amounts is initially a dynamic array that is redimensioned after the user enters the number of cash flow periods.

```
Sub Button_Click
   Dim aprate as Single
   Dim varray() as Double
   Dim cflowper as Integer
   Dim x as Integer
   Dim netpv as Double
   Dim msgtext as string
   cflowper = 2
   ReDim varray(cflowper)
   For x = 1 to cflowper
      varray(x) = 4583
   Next x
   msgtext = "Enter discount rate:"
   aprate = 3.25
   If aprate > 1 then
      aprate = aprate / 100
   End If
   netpv = NPV(aprate,varray())
      msgtext = "The Net Present Value is: " (netpv, "Currency")
End Sub
```

See Also

"Dim Statement" on page 161
"Global Statement" on page 249
"Option Base Statement" on page 347
"Static Statement" on page 460

RefineQuery Method

This method refines a query on a Siebel business component after the query has been executed. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Rem Statement

This standard VB statement identifies a line of code as a comment in a Basic program.

Syntax Rem *comment*

Returns Not applicable

Usage Everything from Rem to the end of the line is ignored when the program is executed.

The single quote (') can also be used to initiate a comment.

Example This program is attached to a button on the Account Form applet that counts the number of corresponding child Contact records.

```
Sub Button1_Click

   Dim i as Integer
   Dim icount as Integer
   Dim oBC as BusComp

   Rem Test this from the Account Contacts View
   Rem This code presumes that Account is the parent BusComp
   Rem BusObject returns the business object
   Rem associated with a control or applet.

   Rem GetBusComp here returns a reference
   Rem to the BC that is in the UI context.

   set oBC = me.BusObject.GetBusComp("Contact")

   Rem FirstRecord positions you at the
   Rem first record in the business component.
   Rem FirstRecord, NextRecord, and so on, do not return Booleans.
   Rem Siebel VB does not have a Boolean data type.

   i = oBC.FirstRecord Rem Returns 0 if fails, 1 if succeeds
   if i <> 1 then
```

```
else
      icount = 0
      Rem This is a sample of using a while statement to loop.
      Rem NextRecord returns 1 if it succesfully
      Rem moved to the next record in the BC

      While i = 1
         icount = icount + 1
         i = oBC.NextRecord   Rem Returns 1 if successful
      wend
      oBC.FirstRecord
      end if

End Sub
```

RemoveChild() Method

RemoveChild removes a child property set from a parent property set. For details, read *Siebel Object Interfaces Reference*.

RemoveProperty Method

RemoveProperty removes a property from a business service or a property set. For details, read *Siebel Object Interfaces Reference*.

Reset Method

This method removes properties and child property sets from a property set. For details, read *Siebel Object Interfaces Reference*.

Reset Statement

This standard VB statement closes every open disk file and writes to disk any data in the operating system buffers.

Syntax Reset

Argument	Description
Not applicable	

Returns Not applicable

Example This example creates a file, puts the numbers 1 through 10 in it, and then attempts to get past the end of the file. The On Error statement traps the error, and execution goes to the Debugger code, which uses Reset to close the file before exiting.

```
Sub Button_Click
' Put the numbers 1-10 into a file
   Dim x as Integer
   Dim y as Integer
   On Error Goto Debugger
   Open "c:\temp001" as #1 Len = 2
   For x = 1 to 10
      Put #1,x, x
   Next x
   Close #1
   msgtext = "The contents of the file is:" & Chr(10)
   Open "C:\TEMP001" as #1 Len = 2
   For x = 1 to 10
      Get #1,x, y
      msgtext = msgtext & Chr(10) & y
   Next x
done:
   Close #1
   Kill "c:\temp001"
   Exit Sub

Debugger:
   TheApplication.RaiseErrorText "Error " & Err & " occurred.
Closing open file."
```

```
      Reset
         Resume done
   End Sub
```

See Also "Close Statement" on page 129

Resume Statement

This standard VB statement halts an error-handling routine.

Syntax A Resume Next

Syntax B Resume *label*

Syntax C Resume [0]

Argument	Description
label	The label that identifies the program line to go to after handling an error

Returns Not applicable

Usage When the Resume Next statement is used, control is passed to the statement that immediately follows the statement in which the error occurred.

When the Resume [0] statement is used, control is passed to the statement in which the error occurred.

The location of the error handler that has caught the error determines where execution resumes. If an error is trapped in the same procedure as the error handler, program execution resumes with the statement that caused the error. If an error is located in a different procedure from the error handler, program control reverts to the statement that last called out the procedure containing the error handler.

See Also "Erl Function" on page 177
"Err Function" on page 179
"Err Statement" on page 180
"Error Function" on page 182
"Error Statement" on page 184
"On Error Statement" on page 341
"Trappable Errors" on page 523

Right Function

This standard VB function returns a portion of a string beginning at the end of the string.

Syntax Right[$](*string*, *length*)

Argument	Description
string	A string or string expression containing the characters to copy
length	The number of characters to copy

Returns A string of length *length* copied from the end of *string*.

Usage If *length* is greater than the length of *string,* Right returns the whole string.

Right accepts any type of *string,* including numeric values, and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

Example This example checks for the extension .bmp in a filename entered by a user and activates the Paintbrush application if the file is found. Note that this uses the Option Compare Text statement to accept either uppercase or lowercase letters for the filename extension.

```
Option Compare Text
Sub Button_Click
   Dim filename as String
   Dim x
   filename ="d:\temp\picture.BMP"
   extension = Right(filename,3)
   If extension = "BMP" then
      x = Shell("PBRUSH.EXE",1)
      Sendkeys "%FO" & filename & "{Enter}", 1
   Else
```

```
        End If
    End Sub
```

See Also

"InStr Function" on page 265
"Left Function" on page 290
"Len Function" on page 292
"LTrim Function" on page 309
"Mid Function" on page 312
"Mid Function" on page 312
"RTrim Function" on page 396
"Trim Function" on page 486

RmDir Statement

This standard VB statement removes a folder.

Syntax RmDir [*drive*:][*folder*\\]*folder*

Argument	Description
drive:	(Optional) The name of the drive from which the folder is to be removed, as a letter, or a string expression representing the drive name
folder\\	If the folder is to be removed from a folder other than the default folder of the specified drive (or the default drive if none is specified), the path to the folder to be removed
folder	The name of the folder to be removed

Returns Not applicable

Usage The folder to be removed must be empty, except for the working (.) and parent (..) folders.

The default folder cannot be removed. To remove the default folder, you must first make another folder current on the drive on which the folder to be removed resides.

Example This example makes a new temporary folder in C:\ and then deletes it.

```
Sub Button_Click
   Dim path as String
   On Error Resume Next
   path = CurDir(C)
   If path <> "C:\" then
      ChDir "C:\"
   End If
   MkDir "C:\TEMP01"
   If Err = 75 then
   Else
      RmDir "C:\TEMP01"
   End If
End Sub
```

See Also

"ChDir Statement" on page 118
"ChDrive Statement" on page 120
"CurDir Function" on page 141
"Dir Function" on page 167
"MkDir Statement" on page 318

Rnd Function

This standard VB function returns a single-precision random number between 0 and 1.

Syntax Rnd[(*number*)]

Argument	Description
number	A numeric expression indicating how the random number is to be generated

Returns A single-precision pseudo-random number between 0 and 1.

Usage If *number* is less than zero, the specified number is used as the seed for a pseudo-random number, which is generated every time the Rnd function is executed. If *number* is greater than zero, or is omitted, Rnd generates a sequence of pseudo-random numbers, in which each execution of the Rnd function uses the next number in the sequence. If *number* is equal to zero, Rnd uses the number most recently generated.

The same sequence of random numbers is generated whenever Rnd is run, unless the random number generator is re-initialized by the Randomize statement.

Example This example generates a random string of characters within a range. The Rnd function is used to set the range between lowercase *a* and *z*. The second For...Next loop is to slow down processing in the first For...Next loop so that Randomize can be seeded with a new value each time from the Timer function.

```
Sub Button_Click
   Dim x as Integer
   Dim y
   Dim str1 as String
   Dim str2 as String
   Dim letter as String
   Dim randomvalue
   Dim upper, lower
   Dim msgtext
   upper = Asc("z")
   lower = Asc("a")
   newline = Chr(10)
```

```
    For x = 1 to 26
       Randomize
       randomvalue = Int(((upper - (lower + 1)) * Rnd) + lower)
       letter = Chr(randomvalue)
       str1 = str1 & letter
       For y = 1 to 1500
       Next y
    Next x
       msgtext = str1
 End Sub
```

See Also

"Exp Function" on page 188
"Fix Function" on page 197
"Int Function" on page 268
"Log Function" on page 303
"Randomize Statement" on page 373
"Sgn Function" on page 430
"Sqr Function" on page 459

Rset Statement

This standard VB function right-aligns one string inside another string.

Syntax Rset *string* = *string-expression*

Placeholder	Description
string	The string to receive the right-aligned characters
string-expression	The string containing the characters to put into *string*

Returns Not applicable

Usage If *string* is longer than *string-expression,* the leftmost characters of *string* are replaced with spaces.

If *string* is shorter than *string-expression,* only the leftmost characters of *string-expression* are copied.

Rset cannot be used to assign variables of different user-defined types.

Example This example uses Rset to right-align an amount entered by the user in a field that is 15 characters long. It then pads the extra spaces with asterisks (*) and adds a dollar sign ($) and decimal places (if necessary).

```
Sub Button_Click

   Dim amount as String * 15
   Dim x as Integer
   Dim msgtext as String
   Dim replacement as String
   Dim position as Integer

   replacement = "*"
   amount = 234.56
   position = InStr(amount,".")
   If position = 0 then
      amount = Rtrim(amount) & ".00"
   End If
   Rset amount = "$" & Rtrim(amount)
   length = 15-Len(Ltrim(amount))
```

```
    For x = 1 to length
       Mid(amount,x) = replacement
       Next x
End Sub
```

See Also "Lset Statement" on page 307

RTrim Function

This standard VB statement copies a string and removes any trailing spaces.

Syntax RTrim[$](*string*)

Argument	Description
string	A string or string expression

Returns A string with any trailing spaces removed.

Usage RTrim accepts any type of *string,* including numeric values, and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

Example For an example, read "Rset Statement" on page 394.

See Also

"Left Function" on page 290
"Len Function" on page 292
"LTrim Function" on page 309
"Mid Function" on page 312
"Mid Statement" on page 314
"Right Function" on page 388
"Trim Function" on page 486

Second Function

This standard VB function returns the second component (0 to 59) of a date-time value.

Syntax Second(*time*)

Argument	Description
time	Any numeric or string expression that can evaluate to a date-time or time value

Returns If the expression evaluates to a date-time or time value, the second component of that value; otherwise 0.

Usage *Time* can be of any type, including strings, and the Second function attempts to convert the input value to a date-time value.

For Second to function without an error, the values passed to it must be in some form that can be interpreted as a time or date-time value. Thus, 13:26:39 or 1:45:12 PM returns valid results, but 1326 returns a 0.

The return value is a variant of vartype 2 (integer). If the value of *time* is NULL, a variant of vartype 1 (Null) is returned.

Example This example displays the last saved date and time for a file whose name is entered by the user.

```
Sub Button_Click
   Dim filename as String
   Dim ftime
   Dim hr, min
   Dim sec
   Dim msgtext as String
i: msgtext = "Enter a filename:"
   filename = "d:\temp\trace.txt"
   If filename = "" then
      Exit Sub
   End If
   On Error Resume Next
   ftime = FileDateTime(filename)
```

```
            If Err <> 0 then
               Goto i:
            End If
            hr = Hour(ftime)
            min = Minute(ftime)
               sec = Second(ftime)
         End Sub
```

See Also

"Date Statement" on page 147
"DateSerial Function" on page 149
"DateValue Function" on page 151
"Day Function" on page 153
"Hour Function" on page 257
"Minute Function" on page 316
"Month Function" on page 320
"Now Function" on page 331
"Time Statement" on page 475
"TimeSerial Function" on page 479
"TimeValue Function" on page 481
"WebApplet_InvokeMethod Event" on page 499
"Year Function" on page 515

Seek Function

This standard VB function returns the current file position for an open file.

Syntax Seek(*filenumber*)

Argument	Description
filenumber	The number assigned to the file to be queried in the Open statement

Returns The position in the file for the next operation.

Usage For files opened in random mode, Seek returns the number of the next record to be read or written. For other modes, Seek returns the file offset for the next operation. The first byte in the file is at offset 1, the second byte is at offset 2, and so on. The return value is a long.

Example This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, `CreateFile`, creates the file `c:\temp001` used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   Rem Put the numbers 10-100 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 10 to 100 step 10
      Write #1, x
   Next x
   Close #1
End Sub

Sub Button_Click
   Dim testscore as String
   Dim x
   Dim y
   Dim newline
   Call CreateFile
```

```
        Open "c:\temp001" for Input as #1
        x = 1
        newline = Chr(10)
        msgtext = "The test scores are: " & newline
        Do Until x = Lof(1)
           Line Input #1, testscore
           x = x + 1
           y = Seek(1)
           If y>Lof(1) then
              x = Lof(1)
           Else
              Seek 1,y
           End If
           msgtext = msgtext & newline & testscore
        Loop
        Close #1
           Kill "c:\temp001"
     End Sub
```

See Also

“Get Statement” on page 218
“Open Statement” on page 344
“Put Statement” on page 367
“Seek Statement” on page 401

Seek Statement

Seek sets the position within an open file for the next read or write operation.

Syntax Seek [#]*filenumber, position*

Argument	Description
filenumber	The number assigned in the Open statement to the file to be queried
position	An expression of type long representing the starting position of the next record number for a random read or write operation, or the byte offset from the beginning of the file

Returns Not applicable

Usage If you write to a file after seeking beyond the end of the file, the file's length is extended. Basic returns an error message if a Seek operation is attempted that specifies a negative or zero position.

For files opened in Random mode, *position* is a record number; for other modes, *position* is a byte offset. *Position* is in the range 1 to 2,147,483,647. The first byte or record in the file is at position 1, the second is at position 2, and so on.

Example This example reads the contents of a sequential file line by line (to a carriage return) and displays the results. The second subprogram, `CreateFile`, creates the file `C:\temp001` used by the main subprogram.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFile

Sub CreateFile
   Rem Put the numbers 10-100 into a file
   Dim x as Integer
   Open "c:\temp001" for Output as #1
   For x = 10 to 100 step 10
      Write #1, x
   Next x
   Close #1
End Sub
```

```
Sub Button_Click
   Dim testscore as String
   Dim x
   Dim y
   Dim newline
   Call CreateFile
   Open "c:\temp001" for Input as #1
   x = 1
   newline = Chr(10)
   msgtext = "The test scores are: " & newline
   Do Until x = Lof(1)
      Line Input #1, testscore
      x = x + 1
      y = Seek(1)
      If y>Lof(1) then
         x = Lof(1)
      Else
         Seek 1,y
      End If
      msgtext = msgtext & newline & testscore
   Loop
   Close #1
      Kill "c:\temp001"
End Sub
```

See Also

"Get Statement" on page 218
"Open Statement" on page 344
"Put Statement" on page 367
"Seek Function" on page 399

Select Case Statement

This standard VB control structure executes one or more statements, depending on the value of an expression.

Syntax

Select Case *testexpression*

Case *expressionList*

[*statement_block*]

[Case *expressionList*

[*statement_block*]]

.

.

[Case Else

[*statement_block*]

End Select

Placeholder	Description
testexpression	Any expression containing a variable to test
expressionList	One or more expressions that contain a possible value for *testexpression*
statement_block	One or more lines of code to execute if *testexpression* equals a value in *expressionList*

Returns Not applicable

Usage When there is a match between *testexpression* and one of the values in *expressionList*, the *statement_block* following the Case clause is executed. When the next Case clause is reached, execution control goes to the statement following the End Select statement.

The *expressionList(s)* can be a comma-separated list of expressions of the following forms:

```
expression

expression To expression

Is comparison_operator expression
```

The type of each *expression* must be compatible with the type of *testexpression*.

Each *statement_block* can contain any number of statements on any number of lines.

NOTE: When the To keyword is used to specify a range of values, the smaller value must appear first. The *comparison_operator* used with the Is keyword is one of the following: `<`, `>`, `=`, `< =`, `> =`, `< >`. You must also use the Is operator when the Case is one end of a range, for example, `Case Is < 100`.

See Also

"If...Then...Else Statement" on page 259
"On...GoTo Statement" on page 340
"Option Compare Statement" on page 350

SendKeys Statement

This standard VB statement sends keystrokes to an active Windows application.

Syntax SendKeys *string*[, *wait*]

Argument	Description
string	A string or string expression containing the characters to send
wait	An integer expression specifying whether to wait until every key is processed before continuing program execution, where: ■ -1 = wait ■ 0 = don't wait

Returns Not applicable

Usage The keystrokes are represented by characters of *string*.

The default value for *wait* is 0 (FALSE).

To specify an alphanumeric character, enter it in *string*. For example, to send the character *a,* use `a` as *string*. Several characters can be combined in one string: If *string* is `abc`, then *a*, *b*, and *c* are sent to the application.

To specify that the SHIFT, ALT, or CTRL key should be pressed simultaneously with a character, prefix the character with

+	to specify SHIFT
%	to specify ALT
^	to specify CTRL

Parentheses can be used to specify that the SHIFT, ALT, or CTRL key should be pressed with a group of characters. For example, `%(abc)`is equivalent to `%a%b%c`.

The following characters must be enclosed in braces if they are to be understood as literal characters by SendKeys; otherwise they have specific meanings as follows:

+	SHIFT key
%	ALT key
^	CTRL key
( )	Apply a shift state to the enclosed characters
~	Newline. Use "~" for the ENTER key on the alphanumeric keyboard, and use "{Enter}" for the ENTER key on the numeric keypad
{ }	Used to make the enclosed characters literals
[ ]	No special meaning for SendKeys, but may have special meaning in other applications

For example, a *string* equal to `{%}` specifies a literal percent character, %.

Use `{{}` to send a left brace and `{}}` to send a right brace.

To send the same key several times, enclose the character in braces and specify the number of keys sent after a space. For example, use `{X 20}` to send 20 *X* characters.

To send one of the nonprintable keys, use a special keyword inside braces:

Key	Keyword
BACKSPACE	`{BACKSPACE}` or `{BKSP}` or `{BS}`
BREAK	`{BREAK}`
CAPS LOCK	`{CAPSLOCK}`
CLEAR	`{CLEAR}`
DELETE	`{DELETE}` or `{DEL}`
DOWN ARROW	`{DOWN}`
END	`{END}`
ENTER (on numeric keypad)	`{ENTER}`

Key	Keyword
ESC	{ESCAPE} or {ESC}
HELP	{HELP}
HOME	{HOME}
INSERT	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}

To send one of the function keys (F1 to F15), simply enclose the name of the key inside braces. For example, to send F5, use `{F5}`.

Note that special keywords can be used in combination with +, %, and ^. For example, `%{TAB}` means ALT + TAB. Also, you can send several special keys in the same way as you would send several normal keys: `{UP 25}` sends 25 up arrows.

SendKeys can send keystrokes only to the currently active application. Therefore, you have to use the AppActivate statement to activate an application before sending keys (unless it is already active).

SendKeys cannot be used to send keys to an application that was not designed to run under Windows.

Example This example starts the Windows Phone Dialer application and dials a phone number entered by the user.

```
Sub Button_Click
   Dim phonenumber, msgtext
   Dim x
   phonenumber = 650-555-1212
   x = Shell ("Terminal.exe",-1)
   SendKeys "%N" & phonenumber & "{Enter}", -1
End Sub
```

See Also "Shell Function" on page 432

Service_InvokeMethod Event

The InvokeMethod event is called after the InvokeMethod method is called on a business service. For details, read *Siebel Object Interfaces Reference*.

Service_PreInvokeMethod Event

The PreInvokeMethod event is called before a specialized method is invoked on the business service. For details, read *Siebel Object Interfaces Reference*.

Set Statement

This standard VB statement assigns a COM object, such as an application, to a variable. Within Siebel Tools, it is used to create an instance of a Siebel object.

Syntax Set *variableName* = *objectExpression*

Argument	Description
variableName	An object variable or variant variable
objectExpression	An expression that evaluates to an object—typically a function, an object member, or Nothing

Returns Not applicable

Usage The following example shows how to use the Set statement:

```
Dim COMObject As Object
Set COMObject = CreateObject("spoly.cpoly")
COMObject.reset
```

NOTE: If you omit the keyword Set when assigning an object variable, Siebel VB tries to copy the default member of one object to the default member of another. This usually results in a run-time error:

```
' Incorrect code - tries to copy default member!
COMObject = GetObject("","spoly.cpoly")
```

Set differs from Let in that Let assigns an expression to a Siebel VB variable. For example,

`Set o1 = o2`	sets the object reference
`Let o1 = o2`	sets the value of the default member

Example This example creates an Opportunity Siebel business component outside the context of the user interface. The program prevents the user from deleting an account if there are opportunities associated with it. For details on the Siebel VB methods and objects used in this example, read *Siebel Object Interfaces Reference*.

```
Function BusComp_PreDeleteRecord As Integer

   Dim iReturn as integer
   Dim oBC as BusComp
   Dim oBO as BusObject
   Dim sAcctRowId as string
   iReturn = ContinueOperation
   sAcctRowId = me.GetFieldValue("Id")

   set oBO = theApplication.GetBusObject("Opportunity")
   set oBC = oBO.GetBusComp("Opportunity")

   With oBC
      .SetViewMode AllView
      .ActivateField "Account Id"
      .ClearToQuery
      .SetSearchSpec "Account Id", sAcctRowId
      .ExecuteQuery ForwardOnly
      if (.FirstRecord) = 1 then
         ‘Opportunities exist for the Account - Delete is not
allowed
         iReturn = CancelOperation
      end if
   End With

   BusComp_PreDeleteRecord = iReturn
   Set oBC = Nothing
      Set oBO = Nothing

End Function
```

See Also

“CreateObject Function” on page 134
“Is Operator” on page 275
“Me” on page 311
“New Operator” on page 325
“Nothing Function” on page 329
“Object Class” on page 336
“Typeof Function” on page 489

SetAttr Statement

This standard VB statement sets the file attributes for a specified file.

Syntax

SetAttr *pathname, attributes*

Argument	Description
pathname	A string or string expression evaluating to the name of the file to modify
attributes	An integer expression containing the new attributes for the file

Returns

Not applicable

Usage

Wildcards are not allowed in *pathname*. If the file is open, you can modify its attributes, but only if it is opened for Read access. Here is a description of attributes that can be modified:

Value	Meaning
0	Normal file
1	Read-only file
2	Hidden file
4	System file
32	Archive—file has changed since last backup

Example

For an example, read "Select Case Statement" on page 403.

See Also

"FileAttr Function" on page 190
"GetAttr Function" on page 222

SetFieldValue Method

SetFieldValue assigns the new value to the named field for the current row of a Siebel business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetFormattedFieldValue Method

SetFormattedFieldValue assigns the new value to the named field for the current row of a Siebel business component. SetFormattedFieldValue accepts the field value in the current local format. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetMultipleFieldValues Method

SetMultipleFieldValues() allows users to set the field values for a particular record as specified in the property set input argument. For details, read *Siebel Object Interfaces Reference*.

SetNamedSearch Method

SetNamedSearch sets a named search specification on a Siebel business component. A named search specification is identified by the *searchName* argument. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetPositionId Method

SetPositionId() changes the position of the current user to the value specified in the input argument. For SetPositionId() to succeed, the user must be assigned to the position to which the user is changing. For details, *Siebel Object Interfaces Reference*.

SetPositionName Method

SetPositionName() changes the position of the current user to the value specified in the input argument. For SetPositionName() to succeed, the user must be assigned to the position to which the user is changing. For details, read *Siebel Object Interfaces Reference*.

SetProfileAttr Method

SetProfileAttr is used in personalization to assign values to attributes in a user profile. For details, read *Siebel Object Interfaces Reference*.

SetProperty Method

This method assigns a value to a property of a business service, property set, or control. For details, read *Siebel Object Interfaces Reference*.

SetSearchExpr Method

SetSearchExpr sets an entire search expression on a Siebel business component, rather than setting one search specification per field. Syntax is similar to that on the Predefined Queries screen. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetSearchSpec Method

SetSearchSpec sets the search specification for a particular field in a Siebel business component. This method must be called before ExecuteQuery. It is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetSharedGlobal Method

The SetSharedGlobal property sets a shared user-defined global variable, which may be accessed using GetSharedGlobal. This method is used with the application object. For details, read *Siebel Object Interfaces Reference*.

SetSortSpec Method

SetSortSpec sets the sorting specification for a query on a Siebel business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetType Method

SetType assigns a data value to a type member of a property set. For details, read *Siebel Object Interfaces Reference*.

SetUserProperty Method

SetUserProperty sets the value of a named Siebel business component user property. The user properties are similar to instance variables of a business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

SetValue Method

The SetValue method sets the contents of a specified control on a Siebel applet to the value indicated. It is also used to assign a data value to a value member of a property set. This method is used with control objects and property sets. For details, read *Siebel Object Interfaces Reference*.

SetViewMode Method

SetViewMode sets the visibility type for a Siebel business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Sgn Function

This standard VB function returns a value indicating the sign of a number.

Syntax Sgn(*number*)

Argument	Description
number	A numeric expression for which the sign is to be determined

Returns If number is less than zero, -1.

If number is equal to zero, 0.

If number is greater than zero, 1.

Example This example tests the value of the variable profit and displays 0 for profit if it is a negative number. The subroutine uses Sgn to determine whether profit is positive, negative, or zero.

```
Sub Button_Click
      Dim profit as Single
      Dim expenses
      Dim sales
      expenses = 100000
      sales = 200000
      profit = Val(sales)-Val(expenses)
      If Sgn(profit) = 1 then
         'Yeah! We turned a profit!
      ElseIf Sgn(profit) = 0 then
         'Okay. We broke even.
      Else
         'Uh, oh. We lost money.
         End If
End Sub
```

See Also

"Exp Function" on page 188
"Fix Function" on page 197
"Int Function" on page 268
"Log Function" on page 303
"Rnd Function" on page 392
"Sqr Function" on page 459

Shell Function

This standard VB function starts a Windows application and returns its task ID.

Syntax Shell(*pathname*, [*windowStyle*])

Argument	Description
pathname	A string or string expression evaluating to the name of the program to execute
windowStyle	An integer indicating how the program's window is to be displayed: ■ 1 if a normal window with focus ■ 2 if a minimized window with focus ■ 3 if a maximized window with focus ■ 4 if a normal window without focus ■ 7 if a minimized window without focus

Returns The task ID for the program, a unique number that identifies the running program.

Usage Shell runs an executable program. *Pathname* can be the name of any valid .COM, .EXE, .BAT, or .PIF file. Arguments and command line switches can be included. If *pathname* is not a valid executable file name, or if Shell cannot start the program, an error message occurs.

If *windowStyle* is not specified, the default of *windowStyle* is 1 (normal window with focus).

Example This example opens Microsoft Excel upon the click of a button. For other examples, read "Right Function" on page 388 and "SendKeys Statement" on page 405.

```
Sub Button1_Click
   Dim i as long
   i = Shell("C:\Program Files\Microsoft
   Office\Office\EXCEL.EXE",1)
End Sub
```

See Also "SendKeys Statement" on page 405

Sin Function

This standard VB function returns the sine of an angle specified in radians.

Syntax

Sin(*number*)

Argument	Description
number	A numeric expression containing a number representing the size of an angle in radians

Returns

The sine of *number.*

Usage

The return value is between -1 and 1. The return value is single precision if the angle is an integer, currency, or single-precision value; double precision for a long, variant, or double-precision value. The angle is specified in radians and can be either positive or negative.

To convert degrees to radians, multiply by (PI/180). The value of PI is 3.14159.

Example

This example finds the height of a building, given the length of the roof and the roof pitch.

```
Sub Button_Click
   Dim height, rooflength, pitch, msgtext As String
   Const PI = 3.14159
   Const conversion = PI/180
   pitch = 35
   pitch = pitch * conversion
   rooflength = 75
   height = Sin(pitch) * rooflength
   msgtext = "The height of the building is "
   msgtext = msgtext & Format(height, "##.##") & " feet."
End Sub
```

See Also

"Atn Function" on page 90
"Cos Function" on page 133
"Tan Function" on page 471
"Derived Trigonometric Functions" on page 527

Space Function

This standard VB function returns a string of spaces.

Syntax Space[$](*number*)

Argument	Description
number	A numeric expression indicating the number of spaces to return

Returns A string of *number* spaces.

Usage *Number* can be any numeric data type, but is rounded to an integer. *Number* must be between 0 and 32,767.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example For an example, read "Oct Function" on page 338.

See Also "Spc Function" on page 435
"String Function" on page 465

Spc Function

This standard VB function prints a specified number of spaces.

Syntax Spc(*number*)

Argument	Description
number	An integer or integer expression indicating the number of spaces to print

Returns A string of spaces in the target of a Print statement.

Usage The Spc function can be used only inside a Print statement.

When the Print statement is used, the Spc function uses the following rules for determining the number of spaces to output:

- If *number* is less than the total line width, Spc outputs *number* spaces.
- If *number* is greater than the total line width, Spc outputs *number* Mod *width* spaces.
- If the difference between the current print position and the output line width (call this difference *x*) is less than *number* or *number* Mod *width*, then Spc skips to the next line and outputs *number* - *x* spaces.

  To set the width of a print line, use the Width statement.

Example This example outputs five spaces and the string ABCD to a file. The five spaces are derived by taking 15 Mod 10, or the remainder of dividing 15 by 10.

```
Sub Button_Click
   Dim str1 as String
   Dim x as String * 10
   str1 = "ABCD"
   Open "C:\temp001" For Output As #1
   Width #1, 10
   Print #1, Spc(15); str1
   Close #1
   Open "C:\TEMP001" as #1 Len = 12
   Get #1, 1,x
```

```
      Close #1
      Kill "C:\temp001"
   End Sub
```

See Also

"Print Statement" on page 364
"Space Function" on page 434
"Tab Function" on page 469
"Width Statement" on page 509

SQLClose Function

This custom VB function disconnects from an ODBC data source connection that was established by SQLOpen.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax SQLClose(*connection*)

Argument	Description
connection	A named argument that must be a long integer, returned by SQLOpen

Returns A variant having one of the following values:

Value	Meaning
0	Successful disconnection
-1	Connection is not valid

Usage For information about named arguments, read "Named Arguments" on page 52 and "Call Statement" on page 112.

If you invoke the ODBC function "SQLClose" with an invalid parameter (for example, SQLClose(0) or a variable parameter without an initialized value), the function responds with the undocumented return code of -2, which indicates an invalid data source connection.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example This example opens the data source named SblTest, gets the names in the ODBC data sources, and closes the connection.

```
Sub Button_Click
'   Declarations

    Dim outputStr As String
    Dim connection As Long
    Dim prompt As Integer
    Dim datasources(1 To 50) As Variant
    Dim retcode As Variant

    prompt = 5
'   Open the data source "SblTest"
    connection = SQLOpen("DSN = SblTest", outputStr, prompt: = 4)

    action1 = 1 ' Get the names of the ODBC data sources
    retcode = SQLGetSchema(connection: = connection,action: _
     = 1,qualifier: = qualifier, ref: = datasources())

'   Close the data source connection
    retcode = SQLClose(connection)

End Sub
```

See Also

“SQLError Function” on page 439
“SQLExecQuery Function” on page 442
“SQLGetSchema Function” on page 445
“SQLOpen Function” on page 448
“SQLRequest Function” on page 451
“SQLRetrieve Function” on page 454
“SQLRetrieveToFile Function” on page 457

SQLError Function

This custom VB function can be used to retrieve detailed information about errors that might have occurred when making an ODBC function call. It returns errors for the last ODBC function and the last connection.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax

SQLError(*destination()*)

Argument	Description
destination	A two-dimensional array of type variant, in which each row contains one error

Returns

Not applicable

Usage

Detailed information for each detected error is returned to the caller in the destination array. Each row of the destination array is filled with information for one error. The elements of each row are filled with the following data:

Element 1:	A character string indicating the ODBC error class/subclass
Element 2:	A numeric value indicating the data source native error code
Element 3:	A text message describing the error

Note that information for more than one error may be returned in the destination array. A 0 in the first element of a row indicates the end of error information in the destination array.

If there are no errors from a previous ODBC function call, then a 0 is returned in the caller's array at (1,1). If the array is not two dimensional or does not provide for the return of the preceding three elements, then an error message is returned in the caller's array at (1,1).

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example

This example forces an error to test the SQLError function.

```
Sub Button_Click
' Declarations
   Dim connection As long
   Dim prompt as integer
   Dim retcode as long
   Dim errors(1 To 10, 1 To 3) as Variant

   ' Open the data source
   connection = SQLOpen("DSN = SVBTESTW;UID=DBA;PWD=SQL"
,outputStr, prompt: = 3)

   ' force an error to test SQLError select a nonexistent table
   retcode = SQLExecQuery(connection: = connection, query: =
"select * from notable ")

   ' Retrieve the detailed error message information into the
   ' errors array
   SQLError destination: = errors
   errCounter = 1
   while errors(errCounter,1) <>0
      errCounter = errCounter + 1
   wend

      retcode = SQLClose(connection)

end sub
```

See Also

“SQLClose Function” on page 437
“SQLExecQuery Function” on page 442
“SQLGetSchema Function” on page 445
“SQLOpen Function” on page 448
“SQLRequest Function” on page 451
“SQLRetrieve Function” on page 454
“SQLRetrieveToFile Function” on page 457

SQLExecQuery Function

This custom VB function executes a SQL statement on a connection established by SQLOpen.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax SQLExecQuery(*connection, query*)

Argument	Description
connection	A long integer returned by SQLOpen
query	A string containing a valid SQL statement

Returns The number of columns in the result set for SQL SELECT statements as a variant.

Select Statement	Return Value
`UPDATE`	The number of rows affected by the statement
`INSERT`	The number of rows affected by the statement
`DELETE`	The number of rows affected by the statement
Other SQL statements	0

Usage If the function is unable to execute the query on the specified data source, or if the connection is invalid, a negative error code is returned.

If SQLExecQuery is called and there are any pending results on that connection, the pending results are replaced by the new results.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example

This example performs a query on the data source.

```
Sub Button_Click
   ' Declarations

    Dim connection As Long
    Dim destination(1 To 50, 1 To 125) As Variant
    Dim retcode As long

    ' open the connection
    connection = SQLOpen("DSN = SblTest",outputStr,prompt: = 3)

    ' Execute the query
    query = "select * from customer"
    retcode = SQLExecQuery(connection,query)

    ' retrieve the first 50 rows with the first 6 columns of
    ' each row into the array destination, omit row numbers and
    ' put column names in the first row of the array

   retcode = SQLRetrieve(connection: = connection, _
      destination: = destination, columnNames: = 1,rowNumbers: _
      = 0,maxRows: = 50, maxColumns: = 6,fetchFirst: = 0)

    ' Get the next 50 rows of from the result set
    retcode = SQLRetrieve(connection: = connection, _
      destination: = destination, columnNames: = 1,rowNumbers: _
         = 0,maxRows: = 50, maxColumns: = 6)

    ' Close the connection
       retcode = SQLClose(connection)

End Sub
```

See Also

"SQLClose Function" on page 437
"SQLError Function" on page 439
"SQLGetSchema Function" on page 445
"SQLOpen Function" on page 448
"SQLRequest Function" on page 451
"SQLRetrieve Function" on page 454
"SQLRetrieveToFile Function" on page 457

SQLGetSchema Function

This custom VB function returns a variety of information, including information on the data sources available, current user ID, names of tables, names and types of table columns, and other data source/database related information.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax

SQLGetSchema *connection, action, qualifier, ref()*

Argument	Description
connection	A long integer returned by SQLOpen
action	An integer value from the following table, specifying what is to be returned to *ref()*
qualifier	A string
ref()	An array of type variant for the results appropriate to the action requested; it must be an array even if it has only one dimension with one element

Returns

A variant whose value depends on the *action* requested, as determined by the values in Table 8. A negative return value indicates an error. A -1 is returned if the requested information cannot be found or if the connection is not valid.

Table 8. Values for Action

Action Value	Returns
1	List of available data sources (dimension of *ref()* is 1)
2	List of databases on the current connection (not supported)
3	List of owners in a database on the current connection (not supported)
4	List of tables on the specified connection
5	List of columns in the table specified by *qualifier* (*ref()* must be two dimensions); returns the column name and SQL data type

Table 8. Values for Action

Action Value	Returns
6	The user ID of the current connection user
7	The name of the current database
8	The name of the data source for the current connection
9	The name of the DBMS the data source uses (for example, DB2)
10	The server name for the data source
11	The terminology used by the data source to refer to owners
12	The terminology used by the data source to refer to a table
13	The terminology used by the data source to refer to a qualifier
14	The terminology used by the data source to refer to a procedure

Usage The destination array must be properly dimensioned to support the action, or an error is returned. Actions 2 and 3 are not currently supported. Action 4 returns every table and does not support the use of the *qualifier*. Not every database product and ODBC driver support every action.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example This example opens the data source named SblTest, gets the names in the ODBC data sources, and closes the connection.

```
Sub Button_Click
   'Declarations

       Dim outputStr As String
       Dim connection As Long
       Dim prompt As Integer
       Dim datasources(1 To 50) As Variant
```

```
        Dim retcode As Variant

        prompt = 5
       'Open the data source "SblTest"
        connection = SQLOpen("DSN=SblTest; UID=SADMIN;
PWD=SADMIN", outputStr,prompt:=4)

        action1 = 1 ' Get the names of the ODBC data sources
        retcode = SQLGetSchema(connection:= connection,action:=
1,qualifier:= qualifier, ref:= datasources())


       'Close the data source connection
        retcode = SQLClose(connection)

       End Sub
```

See Also

"SQLClose Function" on page 437
"SQLError Function" on page 439
"SQLExecQuery Function" on page 442
"SQLOpen Function" on page 448
"SQLRequest Function" on page 451
"SQLRetrieve Function" on page 454
"SQLRetrieveToFile Function" on page 457

SQLOpen Function

This custom VB function establishes a connection to an ODBC data source.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax SQLOpen(*connectString*, [*outputString*][, *prompt*])

Argument	Description
connectString	A string or string variable supplying the information required to connect to a data source, including the data source name, user ID, and password, and any other information required by the driver to make the connection
outputString	A string variable to hold the completed connection string if the connection is successful
prompt	An integer specifying when the driver dialog box is displayed

The following table lists the values for *prompt*. When *prompt* is omitted, 2 is assumed.

Prompt Value	Meaning
1	Driver dialog box is always displayed
2	Driver dialog box is displayed only when the specification is not sufficient to make the connection
3	The same as 2, except that dialog boxes that are not required are unavailable and cannot be modified
4	Driver dialog box is not displayed; if the connection is not successful, an error is returned

Returns A long integer representing a unique connection ID, which can be used with other ODBC functions. The completed connection string is returned in *outputString* if this argument is used. If the connection cannot be established, then an ODBC error with a negative numeric value is returned. Test this value using the instructions in "SQLError Function" on page 439.

Usage The *connectString* variable generally takes the following form; however, it must follow the format dictated by the ODBC driver you are using.

```
"DSN=dataSourceName;UID=loginID;PWD=password"
```

As the example that follows shows, some parts of this string may not be required; you must supply whatever information is required by the ODBC driver to make the connection. For details on the connect string used to access a Siebel application, read Siebel Technical Note #206.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example This example opens the data source named SblTest, gets the names in the ODBC data sources, and closes the connection.

```
Sub Button_Click
   Dim outputStr As String
   Dim connection As Long
   Dim prompt As Integer
   Dim action As Integer
   Dim qualifier As String
   Dim datasources(1 To 50) As Variant
   Dim retcode As Variant

   prompt = 4
   Set ret = TheApplication.NewPropertySet()

   ' Open the datasource "SblTest" with a user name of sa, _
   password of sa
   connection = _
   SQLOpen("DSN=SblTest;UID=sa;PWD=sa",outputStr,prompt:=4)
   action = 1 ' Get the names of the ODBC data sources
```

```
      retcode = SQLGetSchema(connection:=connection, _
         action:=1, _

         qualifier:=qualifier, _
         ref:=datasources())

      ' Close the data source connection
      retcode = SQLClose(connection)
   End Sub
```

See Also

"SQLClose Function" on page 437
"SQLError Function" on page 439
"SQLExecQuery Function" on page 442
"SQLGetSchema Function" on page 445
"SQLRequest Function" on page 451
"SQLRetrieve Function" on page 454
"SQLRetrieveToFile Function" on page 457

SQLRequest Function

This custom VB function establishes a connection to a data source, executes a SQL statement contained in *query$*, returns the results of the request in the *ref()* array, and closes the connection.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax SQLRequest(*connectString, query, outputString*[, *prompt*][, *columnNames*], *ref()*)

Argument	Description
connectString	A string or string variable specifying the data source to connect to. For details on the connect string, read "SQLOpen Function" on page 448.
query	A SQL query
outputString	A string variable to hold the completed connection string if the connection is successful
prompt	An integer that specifies when driver dialog boxes are displayed. For a table of values for *prompt*, read "SQLOpen Function" on page 448.
columnNames	An integer with a value of 0 or nonzero. When *columnNames* is nonzero, column names are returned as the first row of the *ref()* array. If *columnNames* is omitted, the default is 0.
ref()	An array of type variant for the results appropriate to the action requested; it must be an array even if only one dimension with one element

Returns A variant containing a negative-numbered error code if the connection cannot be made, the query is invalid, or another error condition occurs. If the request is successful, returns a positive number representing the number of results returned or rows affected. Other SQL statements return 0.

Usage The SQLRequest function establishes a connection to the data source specified in *connectString,* executes the SQL statement contained in *query,* returns the results of the request in the *ref()* array, and closes the connection.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example

```
Function WebApplet_PreInvokeMethod (MethodName As String) As Integer
   If MethodName = "queryExtSys" Then

      ' The following opens the datasource SVBTESTW and
      ' executes the query specified by query and returns the
      ' results in destination.

      Dim errors(1 To 10, 1 To 3) As Variant
      Dim destination(1 To 50, 1 To 125)  As Variant
      Dim prompt As Integer
      Dim outputStr As String
      Dim retCode As Integer

      ' In the event of a connection error, do not display a
      ' dialog box, return an error
      prompt = 4

      ' SQL Statement to submit. In this example we'll perform a
      ' simple select
      query = "SELECT * FROM authors"

      ' Invoke the SQLRequest function to submit the SQL, execute the
      ' query and return a result set.
      retCode = SQLRequest("DSN=SVBTESTW;UID=sa;PWD=sa", _
               query, outputStr, prompt, 0, destination())

      ' If retCode < 0,  an error has occurred. Retrieve the first
      ’ error returned in the array and display to the user.
      If retCode < 0 Then
         SQLError destination := errors
         errCounter = 1
```

```
         While errors(errCounter,1) <> 0
            TheApplication.RaiseErrorText "Error  " & _
            "  ODBC error: " & destination(errCounter,1) & _
            "  Numeric code = " & destination(errCounter,2) & _
            "  Error Text = " & destination(errCounter,3)

            errCounter = errCounter + 1
         Wend
      Else
         ' do some processing of the results
      End If

      WebApplet_PreInvokeMethod = CancelOperation
   Else
      WebApplet_PreInvokeMethod = ContinueOperation
   End If

End Function
```

See Also

"SQLClose Function" on page 437
"SQLError Function" on page 439
"SQLExecQuery Function" on page 442
"SQLGetSchema Function" on page 445
"SQLOpen Function" on page 448
"SQLRetrieve Function" on page 454
"SQLRetrieveToFile Function" on page 457

SQLRetrieve Function

This custom VB function fetches the results of a pending query on the connection specified by *connection* and returns the results in the *destination()* array.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax SQLRetrieve(*connection, destination()*[, *maxColumns*][, *maxRows*][, *columnNames*][, *rowNumbers*][, *fetchFirst*])

Argument	Description
connection	The long integer returned by the SQLOpen function
destination()	A two-dimensional array of type variant. The first index of the array cannot exceed 100.
maxColumns	The number of columns to be retrieved in the request; defaults to 0 if this parameter is not used
maxRows	The number of rows to be retrieved in the request; if this argument is not supplied, 0 is assumed
columnNames	An integer with a value of 0 or nonzero. When *columnNames* is nonzero, column names are returned as the first row of the *ref()* array. If *columnNames* is omitted, the default is 0.
rowNumbers	An integer with a value of 0 or nonzero. When *rowNumbers* is nonzero, row numbers are returned as the first row of the *ref()* array. If *rowNumbers* is omitted, the default is 0.
fetchFirst	A positive integer value that causes the result set to be repositioned to the first row of the database, if the database supports this action; returns -1 if this cannot be accomplished

Returns A variant containing the following values:

Result	Returns
Success	The number of rows in the result set or the *maxRows* requested
Unable to retrieve results, or no results pending	-1
No data found by the query	0

Usage If *maxColumns* or *maxRows* is omitted, the array size is used to determine the maximum number of columns and rows retrieved, and an attempt is made to return the entire result set. Extra rows can be retrieved by using SQLRetrieve again and by setting *fetchFirst* to 0. If *maxColumns* specifies fewer columns than are available in the result, SQLRetrieve discards the rightmost result columns until the results fit the specified size.

When *columnNames* is nonzero, the first row of the array is set to the column names as they are specified by the database schema. When r*owNumbers* is nonzero, row numbers are returned in the first column of *destination()*. SQLRetrieve clears the user's array prior to fetching the results.

When *fetchFirst* is nonzero, it causes the result set to be repositioned to the first row if the database supports the function. If the database does not support repositioning, the result set -1 error is returned.

If there are more rows in the result set than can be contained in the *destination()* array or than have been requested using *maxRows*, the user can make repeated calls to SQLRetrieve until the return value is 0.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example This example retrieves information from a data source.

```
Sub Button_Click
' Declarations

    Dim connection As Long
    Dim destination(1 To 50, 1 To 125) As Variant
    Dim retcode As long

   ' open the connection
   connection = SQLOpen("DSN = SblTest",outputStr,prompt: = 3)

   ' Execute the query
   query = "select * from customer"
   retcode = SQLExecQuery(connection,query)

   ' retrieve the first 50 rows with the first 6 columns of
   ' each row into the array destination, omit row numbers and
   ' put column names in the first row of the array

   retcode = SQLRetrieve(connection: = connection, _
      destination: = destination, columnNames: = 1, _
      rowNumbers: = 0, maxRows: = 50, maxColumns: = 6, _
      fetchFirst: = 0)

   ' Get the next 50 rows of from the result set
   retcode = SQLRetrieve(connection: = connection, _
      destination: = destination, columnNames: = 1, _
      rowNumbers: = 0, maxRows: = 50, maxColumns: = 6)

   ' Close the connection
      retcode = SQLClose(connection)
End Sub
```

See Also

"SQLClose Function" on page 437
"SQLError Function" on page 439
"SQLExecQuery Function" on page 442
"SQLGetSchema Function" on page 445
"SQLOpen Function" on page 448
"SQLRequest Function" on page 451
"SQLRetrieveToFile Function" on page 457

SQLRetrieveToFile Function

This custom VB function fetches the results of a pending query on the connection specified by *connection* and stores them in the file specified by *destination*.

NOTE: Siebel VB SQLxxxx functions are supported for non-Unicode databases only.

Syntax SQLRetrieveToFile(*connection*, *destination*[, *columnNames*][, *columnDelimiter*])

Argument	Description
connection	The number returned by the SQLOpen function
destination	A string or string variable containing the filename and path to be used for storing the results
columnNames	One of the following values: ■ nonzero = The first row contains the column headers as specified by the database schema ■ 0 = The column headers are not retrieved The default is 0.
columnDelimiter	The string to be used to delimit the fields in a row; if omitted, a Tab character is used

Returns If successful, a variant containing the number of rows in the result set; if unsuccessful, -1.

Usage The arguments must be named arguments. For information about named arguments, read "Named Arguments" on page 52 and "Call Statement" on page 112.

CAUTION: This function, as well as the other SQL functions available in Siebel VB, should not be used to query the underlying database. Use the Siebel Object Interfaces to query Siebel data. Use the SQL functions only to query non-Siebel data.

Example This example opens a connection to a data source and retrieves information to a file.

```
Sub Button_Click
   'Declarations

   Dim connection As Long
   Dim destination(1 To 50, 1 To 125) As Variant
   Dim retcode As long

   'open the connection

   connection = SQLOpen("DSN = SblTest",outputStr,prompt: = 3)

   ' Execute the query

   query = "select * from customer"
   retcode = SQLExecQuery(connection,query)

   'Place the results of the previous query in the file
   'named by filename and put the column names in the file
   'as the first row.
   'The field delimiter is %

      filename = "c:\myfile.txt"
      columnDelimiter = "%"
      retcode = SQLRetrieveToFile(connection: = connection, _
      destination: = filename, columnNames: = 1, _
      columnDelimiter: = columnDelimiter)

      retcode = SQLClose(connection)

End Sub
```

See Also

"SQLClose Function" on page 437
"SQLError Function" on page 439
"SQLExecQuery Function" on page 442
"SQLGetSchema Function" on page 445
"SQLOpen Function" on page 448
"SQLRequest Function" on page 451
"SQLRetrieve Function" on page 454

Sqr Function

This standard VB function returns the square root of a number.

Syntax Sqr(*number*)

Argument	Description
number	An expression containing the number whose square root is to be found

Returns The square root of *number*.

Usage The return value is single precision for an integer, currency, or single-precision numeric expression; double precision for a long, variant, or double-precision numeric expression.

Example For an example that calculates the square root of 2 as a double-precision floating-point value and displays it in scientific notation, read "Format Function" on page 202.

See Also
"Exp Function" on page 188
"Fix Function" on page 197
"Int Function" on page 268
"Log Function" on page 303
"Rnd Function" on page 392
"Sgn Function" on page 430

Static Statement

This standard VB statement declares variables and allocates storage space.

Syntax Static *variableName* [As *type*] [,*variableName* [As *type*]] ...

Argument	Description
variableName	The name of the variable to declare as static
type	The data type of the variable; if not specified, the type is variant

Returns Not applicable

Usage Variables declared with the Static statement retain their value as long as the program is running. The syntax of Static is exactly the same as the syntax of the Dim statement.

Variables of a procedure can be made static by using the Static keyword in a definition of that procedure. For more information, read "Function...End Function Statement" on page 213 and "Sub...End Sub Statement" on page 467.

See Also
"Dim Statement" on page 161
"Function...End Function Statement" on page 213
"Global Statement" on page 249
"Option Base Statement" on page 347
"ReDim Statement" on page 377
"Sub...End Sub Statement" on page 467

Stop Statement

This standard VB statement halts program execution.

Syntax Stop

Argument	Description
Not applicable	

Returns Not applicable

Usage Stop statements can be placed anywhere in a program to suspend its execution. Although the Stop statement halts program execution, it does not close files or clear variables.

Example This example stops program execution at the user's request.

```
Sub Button_Click
   Dim str1
   str1 = Y
   If str1 = "Y" or str1 = "y" then
      Stop
   End If
End Sub
```

Str Function

This standard VB function returns a string representation of a number.

Syntax

Str[$](*number*)

Argument	Description
number	The number to be represented as a string

Returns

A string representation of *number*.

Usage

The precision in the returned string is single precision for an integer or single-precision numeric expression; double precision for a long or double-precision numeric expression, and currency precision for currency. Variants return the precision of their underlying vartype.

The dollar sign ($) in the function name is optional. If it is specified, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example

This example prompts for two numbers, adds them, and then shows them as a concatenated string.

```
Sub Button_Click
   Dim x as Integer
   Dim y as Integer
   Dim str1 as String
   Dim value1 as Integer
   x = 1
   y = 2
   str1 = "The sum of these numbers is: " & x+y
   str1 = Str(x) & Str(y)
End Sub
```

See Also

"Format Function" on page 202
"Val Function" on page 495

StrComp Function

This standard VB function compares two strings and returns an integer specifying the result of the comparison.

Syntax

StrComp(*string1*, *string2*[, *compare*])

Argument	Description
string1	An expression containing the first string to compare
string2	An expression containing the second string to compare
compare	An integer indicating the method of comparison, where: 0 = case-sensitive 1 = case-insensitive

Returns

One of the following values:

Value	Meaning
-1	*string1* < *string2*
0	*string1* = *string2*
> 1	*string1* > *string2*
Null	*string1* = Null or *string2* = Null

Usage

If *compare* is 0, a case-sensitive comparison based on the ANSI character set sequence is performed. If *compare* is 1, a case-insensitive comparison is done based upon the relative order of characters as determined by the country code setting for your computer. If this argument is omitted, the module-level default, as specified with Option Compare, is used.

The *string1* and *string2* arguments are both passed as variants. Therefore, any type of expression is supported. Numbers are automatically converted to strings.

Example

This example compares a user-entered string to the string `Smith`.

```
Option Compare Text
Sub Button_Click
   Dim lastname as String
   Dim smith as String
   Dim x as Integer
   smith = "Smith"
   lastname = "smith"
   x = StrComp(lastname,smith,1)
   If x = 0 then
      ‘You typed Smith or smith
      End If
End Sub
```

See Also “InStr Function” on page 265
“Option Compare Statement” on page 350

String Function

This standard VB function returns a string consisting of a repeated character.

Syntax A String[$](*number, character*)

Syntax B String[$] (*number, stringExpression*)

Argument	Description
number	The length of the string to be returned
character	An integer or integer expression containing the ANSI code of the character to use
stringExpression	A string argument, the first character of which becomes the repeated character

Returns A string containing *number* repetitions of the specified character.

Usage *Number* must be between 0 and 32,767.

Character must evaluate to an integer between 0 and 255.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example This example places asterisks (*) in front of a string that is printed as a payment amount.

```
Sub Button_Click
   Dim str1 as String
   Dim size as Integer
i: str1 = 666655.23
   If Instr(str1,".") = 0 then
      str1 = str1 + ".00"
   End If
   If Len(str1)>10 then
      Goto i
   End If
   size = 10-Len(str1)
```

```
'Print amount in a space on a check allotted for 10 characters
   str1 = String(size,Asc("*")) & str1
End Sub
```

See Also "Space Function" on page 434
"Str Function" on page 462

Sub...End Sub Statement

This standard VB construct defines a subprogram procedure.

Syntax [Static] [Private] Sub *name* [([Optional] *parameter* [As *type*] , ...)]
End Sub

Argument	Description
name	The name of the subprogram
parameter	A list of parameter names, separated by commas
type	The data type for *parameter*

Returns Not applicable

Usage A call to a subprogram stands alone as a separate statement (read “Call Statement” on page 112). Recursion is supported.

The data type of a parameter can be specified by using a type character or by using the As clause. Record parameters are declared by using an As clause and a type that has previously been defined using the Type statement. Array parameters are indicated by using empty parentheses after the *parameter*. The array dimensions are not specified in the Sub statement. Every reference to an array within the body of the subprogram must have a consistent number of dimensions.

If a *parameter* is declared as optional, its value can be omitted when the function is called. Only variant parameters can be declared as optional, and optional parameters must appear after the required parameters in the Sub statement. To check whether an optional parameter was omitted by the user, use the IsMissing function (read “IsMissing Function” on page 279). For more information on using named parameters, read “Named Arguments” on page 52 and “Call Statement” on page 112.

The procedure returns to the caller when the End Sub statement is reached or when an Exit Sub statement is executed.

The Static keyword specifies that the variables declared within the subprogram retains their values as long as the program is running, regardless of the way the variables are declared.

The Private keyword specifies that the procedures are not accessible to functions and subprograms from other modules. Only procedures defined in the same module have access to a Private subprogram.

Basic procedures use the call by reference convention. This means that if a procedure assigns a value to a parameter, it modifies the variable passed by the caller.

Use Function rather than Sub (read "Function...End Function Statement" on page 213) to define a procedure that has a return value.

CAUTION: You cannot write your own functions or subprograms directly in the methods and events exposed in Siebel Tools. You can write functions and subprograms in the (general) (declarations) section of a given method script. However, if you want your routines to be available throughout the program, you can use the Application_PreInvokeMethod or an external DLL file as a central place to write them. For details, read Siebel Technical Notes #207 and #217.

If you create more than one function or subprogram in the (general) (declarations) section, be sure that any function or subprogram that may be called by other user-defined functions and subprograms appears before the procedure that calls it. Otherwise, you can not compile your procedures.

Example

This example is a subroutine that uses the Sub...End Sub statement.

```
Sub Button1_Click
    'Hello, World.
End Sub
```

See Also

"BusComp Method" on page 92
"Dim Statement" on page 161
"Function...End Function Statement" on page 213
"Global Statement" on page 249
"Option Explicit Statement" on page 352
"Static Statement" on page 460

Tab Function

This standard VB function moves the current print position to the column specified.

Syntax Tab(*position*)

Argument	Description
position	The position at which printing is to occur

Returns Not applicable

Usage The Tab function can be used only inside a Print statement. The leftmost print position is position number 1.

When the Print statement is used, the Tab function uses the following rules for determining the next print position:

- If *position* is less than the total line width, the new print position is *position*.
- If *position* is greater than the total line width, the new print position is *n* Mod *width*.
- If the current print position is greater than *position* or *position* Mod *width*, Tab skips to the next line and sets the print position to *position* or *position* Mod *width*.

To set the width of a print line, use the Width statement.

Example This example prints the octal values for the numbers from 1 to 25. It uses Tab to put five character spaces between the values.

```
Sub Button_Click
      Dim x As Integer
      Dim y As String
      For x = 1 to 25
         y = Oct$(x)
         Print x Tab(10) y
      Next x
End Sub
```

See Also

"Print Statement" on page 364
"Space Function" on page 434
"Spc Function" on page 435
"Width Statement" on page 509

Tan Function

This standard VB function returns the tangent of an angle in radians.

Syntax Tan(*number*)

Argument	Description
number	A numeric expression containing the number of radians in the angle whose tangent is to be returned

Returns The tangent of *number*.

Usage *Number* is specified in radians and can be either positive or negative.

The return value is single precision if the angle is an integer, currency, or single-precision value; double precision for a long, variant, or double-precision value.

To convert degrees to radians, multiply by PI/180. The value of PI is 3.14159.

Example This example finds the height of the exterior wall of a building, given its roof pitch and the length of the building.

```
Sub Button_Click
   Dim bldglen, wallht
   Dim pitch
   Dim msgtext
   Const PI = 3.14159
   Const conversion = PI/180
   On Error Resume Next
   pitch = 35
   pitch = pitch * conversion
   bldglen = 150
   wallht = Tan(pitch) * (bldglen/2)
End Sub
```

See Also "Atn Function" on page 90
"Cos Function" on page 133
"Sin Function" on page 433
"Derived Trigonometric Functions" on page 527

TheApplication Method

TheApplication is a global Siebel method that returns the unique object of type Application. This is the root of every object within the Siebel Applications object hierarchy. Use this method to determine the object reference of the application, which is later used to find other objects, or to invoke methods on the application object. For details, read *Siebel Object Interfaces Reference*.

Time Function

This standard VB function returns a string representing the current time.

Syntax Time[$]

Argument	Description
Not applicable	

Returns An eight-character string of the format *hh:mm:ss,* where *hh* is the hour, *mm* is the minutes, and *ss* is the seconds. The hour is specified in military style and ranges from 0 to 23.

Usage The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function returns a variant of vartype 8 (string).

Example This example writes data to a file if it has not been saved within the last two minutes.

```
Sub Button_Click
   Dim tempfile
   Dim filetime, curtime
   Dim msgtext
   Dim acctno(100) as Single
   Dim x, I
   tempfile = "c:\temp001"
   Open tempfile For Output As #1
   filetime = FileDateTime(tempfile)
   x = 1
   I = 1
   acctno(x) = 0
   Do
      curtime = Time
      acctno(x) = 44
      If acctno(x) = 99 then
         For I = 1 to x -1
            Write #1, acctno(I)
         Next I
         Exit Do
      ElseIf (Minute(filetime) + 2)< = Minute(curtime) then
```

```
            For I = I to x
               Write #1, acctno(I)
            Next I
         End If
         x = x + 1
      Loop
      Close #1
      x = 1
      msgtext = "Contents of c:\temp001 is:" & Chr(10)
      Open tempfile for Input as #1
      Do While Eof(1) <> -1
         Input #1, acctno(x)
         msgtext = msgtext & Chr(10) & acctno(x)
         x = x + 1
      Loop
      Close #1
         Kill "c:\temp001"
   End Sub
```

See Also

"Date Function" on page 146
"Date Statement" on page 147
"Time Statement" on page 475
"Timer Function" on page 477
"TimeSerial Function" on page 479
"TimeValue Function" on page 481

Time Statement

This standard VB statement sets the computer's time.

Syntax Time[$] = *expression*

Placeholder	Meaning
expression	An expression that evaluates to a valid time

Returns Not applicable

Usage When Time (with the dollar sign, $) is used, *expression* must evaluate to a string of one of the following forms:

`hh`	Sets the time to *hh* hours 0 minutes and 0 seconds.
`hh:mm`	Sets the time to *hh* hours *mm* minutes and 0 seconds.
`hh:mm:ss`	Sets the time to *hh* hours *mm* minutes and *ss* seconds.

Time uses a 24-hour clock. Thus, 6:00 P.M. must be entered as 18:00:00.

If the dollar sign ($) is omitted, *expression* can be a string containing a valid date, or a variant of vartype 7 (date) or 8 (string).

If *expression* is not already a variant of vartype 7 (date), Time attempts to convert it to a valid time. It recognizes time separator characters defined in the International section of the Windows Control Panel. Time (without the $) accepts both 12- and 24-hour clocks.

Example This example changes the time on the computer's clock.

```
Sub Button_Click
   Dim newtime As String
   Dim answer As String
   On Error Resume Next
i:
   newtime = "5:30"
   answer = PM
   If answer = "PM" or answer = "pm" then
```

```
            newtime = newtime &"PM"
         End If
         Time = newtime
         If Err <> 0 then
            Err = 0
            Goto i
            End If
      End Sub
```

See Also

“Date Function” on page 146
“Date Statement” on page 147
“Time Function” on page 473
“TimeSerial Function” on page 479
“TimeValue Function” on page 481

Timer Function

This standard VB function returns the number of seconds that have elapsed since midnight.

Syntax Timer

Argument	Description
Not applicable	

Returns The number of seconds that have elapsed since midnight.

Usage The Timer function can be used in conjunction with the Randomize statement to seed the random number generator.

Example This example uses the Timer to find Megabucks numbers.

```
Sub Button_Click
   Dim msgtext As String
   Dim value(9) As Single
   Dim nextvalue As Integer
   Dim x As Integer
   Dim y As Integer

   msgtext = "Your Megabucks numbers are: "
   For x = 1 to 8
      Do
         value(x) = Timer
         value(x) = value(x) * 100
         value(x) = Str(value(x))
         value(x) = Val(Right(value(x),2))
      Loop Until value(x)>1 and value(x)<36
      For y = 1 to 1500
      Next y
   Next x

   For y = 1 to 8
   For x = 1 to 8
      If y <> x then
         If value(y) = value(x) then
            value(x) = value(x) + 1
```

```
                End If
             End If
           Next x
          Next y
          For x = 1 to 8
             msgtext = msgtext & value(x) & " "
             Next x
       End Sub
```

See Also “Randomize Statement” on page 373

TimeSerial Function

This standard VB function returns a time as a variant of type 7 (date/time) for a specific hour, minute, and second.

Syntax TimeSerial(*hour*, *minute*, *second*)

Argument	Description
hour	A numeric expression containing a value from 0 to 23 representing an hour
minute	A numeric expression containing a value from 0 to 59 representing a minute
second	A numeric expression containing a value from 0 to 59 representing a second

Returns A time as a specific hour, minute, and second.

Usage You also can specify relative times for each argument by using a numeric expression representing the number of hours, minutes, or seconds before or after a certain time.

Example This example displays the current time using TimeSerial.

```
Sub Button_Click
   Dim y As Variant
   Dim msgtext As String
   Dim nowhr As Integer
   Dim nowmin As Integer
   Dim nowsec As Integer
   nowhr = Hour(Now)
   nowmin = Minute(Now)
   nowsec = Second(Now)
   y = TimeSerial(nowhr,nowmin,nowsec)
   msgtext = "The time is: " & y
End Sub
```

See Also

"DateSerial Function" on page 149
"DateValue Function" on page 151
"Hour Function" on page 257
"Minute Function" on page 316
"Now Function" on page 331
"Second Function" on page 397
"TimeValue Function" on page 481

TimeValue Function

This standard VB function returns a time value for a specified string.

Syntax TimeValue(*time*)

Argument	Description
time	A string representing a valid date-time value

Returns A date/time value for the time represented by *time*.

Usage The TimeValue function returns a variant of vartype 7 (date/time) that represents a time between 0:00:00 and 23:59:59, or 12:00:00 A.M. and 11:59:59 P.M., inclusive.

Example This example writes a variable to a disk file based on a comparison of its last saved time and the current time. Note that the variables used for the TimeValue function are dimensioned as double, so that calculations based on their values work properly.

```
Sub Button_Click
   Dim tempfile As String
   Dim ftime As Variant
   Dim filetime as Double
   Dim curtime as Double
   Dim minutes as Double
   Dim acctno(100) as Integer
   Dim x, I
   tempfile = "C:\TEMP001"
   Open tempfile For Output As 1
   ftime = FileDateTime(tempfile)
   filetime = TimeValue(ftime)
   minutes = TimeValue("00:02:00")
   x = 1
   I = 1
   acctno(x) = 0
   Do
      curtime = TimeValue(Time)
      acctno(x) = 46
      If acctno(x) = 99 then
         For I = I to x-1
```

```
            Write #1, acctno(I)
         Next I
         Exit Do
      ElseIf filetime + minutes< = curtime then
         For I = I to x
            Write #1, acctno(I)
         Next I
      End If
      x = x + 1
   Loop
   Close #1
   x = 1
   msgtext = "You entered:" & Chr(10)
   Open tempfile for Input as #1
   Do While Eof(1) <> -1
      Input #1, acctno(x)
      msgtext = msgtext & Chr(10) & acctno(x)
      x = x + 1
   Loop
   Close #1
      Kill "C:\TEMP001"
End Sub
```

See Also

"DateSerial Function" on page 149
"DateValue Function" on page 151
"Hour Function" on page 257
"Minute Function" on page 316
"Now Function" on page 331
"Second Function" on page 397
"TimeSerial Function" on page 479

Trace Method

The Trace method appends a message to the trace file. Trace is useful for debugging SQL query execution. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

TraceOff Method

TraceOff turns off the tracing started by the TraceOn method. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

TraceOn Method

TraceOn turns on the tracking of allocations and de-allocations of Siebel objects and SQL statements generated by the Siebel application. It is used with the application object. For details, read *Siebel Object Interfaces Reference*.

Trim Function

This standard VB function returns a copy of a string after removing leading and trailing spaces.

Syntax Trim[$](*string*)

Argument	Description
string	A literal or expression from which leading and trailing spaces are to be removed

Returns A copy of *string* with leading and trailing spaces removed.

Usage Trim accepts expressions of type string. Trim accepts any type of *string*, including numeric values, and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function typically returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

See Also "Left Function" on page 290
"Len Function" on page 292
"LTrim Function" on page 309
"Mid Function" on page 312
"Mid Statement" on page 314
"Right Function" on page 388
"RTrim Function" on page 396

Type Statement

This standard VB statement declares a user-defined type.

Syntax Type *userType*

field1 As *type1*

field2 As *type2*

...
End Type

Argument	Description
userType	The name of the user-defined type
field1, field2	The names of fields in the user-defined type
type1, type2	The data types of the respective fields

Returns Not applicable

Usage The user-defined type declared by Type can be used in a Dim statement to declare a record variable. A user defined type is sometimes referred to as a *record type* or a *structure type*.

Field cannot be an array. However, arrays of records are allowed.

The Type statement is not valid inside a procedure definition. It must be placed in the (general) (declarations) section, shown in Figure 1 on page 348. User defined Types cannot be passed to COM Object functions or subroutines.

To access the fields of a record, use syntax of the form:

```
recordName.fieldName
```

To access the fields of an array of records, use syntax of the form:

```
arrayName(index).fieldName
```

No memory is allocated when a type is defined. Memory is allocated when a variable of the user defined type is declared with a Dim statement. Declaring a variable of a user defined type is called *instantiating*, or *declaring an instance of*, the type.

Example

This example shows a Type and Dim statement for a record. You must define a record type before you can declare a record variable. The subroutine then references a field within the record.

```
Type Testrecord
   Custno As Integer
   Custname As String
End Type

Sub Button_Click
   Dim myrecord As Testrecord
   Dim msgText As String
i:
   myrecord.custname = "Chris Smith"
   If myrecord.custname = "" then
      Exit Sub
   End If
End Sub
```

See Also

"Deftype Statement" on page 158
"Dim Statement" on page 161

Typeof Function

This standard VB function returns a value indicating whether an object is of a given class.

Syntax If Typeof *objectVariable* Is *className* Then...

Placeholder	Meaning
objectVariable	The object to be tested
className	The class to which the object is to be compared

Returns The Typeof function returns -1 if the object is of the specified type, 0 if it is not.

Usage Typeof can be used only in an If statement and cannot be combined with other Boolean operators. That is, Typeof can be used only exactly as shown in the syntax line.

To test whether an object does *not* belong to a class, use the following code structure:

```
If Typeof objectVariable Is className Then

[Perform some action.]
Else
   [Perform some action.]
End If
```

See Also “CreateObject Function” on page 134
“GetObject Function” on page 234
“Is Operator” on page 275
“Me” on page 311
“New Operator” on page 325
“Nothing Function” on page 329
“Object Class” on page 336

UBound Function

This standard VB function returns the upper bound of the subscript range for the specified array.

Syntax UBound(*arrayName*[, *dimension*])

Argument	Description
arrayName	The variable name of the array to be tested
dimension	The array dimension whose upper bound is to be returned

Returns The upper bound of the subscript range for the specified dimension of the specified array.

Usage The dimensions of an array are numbered starting with 1. If *dimension* is not specified, 1 is used as a default.

LBound can be used with UBound to determine the length of an array.

Example This example resizes an array if the user enters more data than can fit in the array. It uses LBound and UBound to determine the existing size of the array and ReDim to resize it. Option Base sets the default lower bound of the array to 1.

```
Option Base 1

Sub Button_Click
   Dim arrayvar() as Integer
   Dim count as Integer
   Dim answer as String
   Dim x, y as Integer
   Dim total
   total = 0
   x = 1
   count = 2
   ReDim arrayvar(count)
start:
   Do until x = count + 1
      arrayvar(x) = 88
      x = x + 1
```

```
    Loop
       x = LBound(arrayvar,1)
    count = UBound(arrayvar,1)
    For y = x to count
          total = total + arrayvar(y)
       Next y
End Sub
```

See Also

"Dim Statement" on page 161
"Global Statement" on page 249
"LBound Function" on page 287
"Option Base Statement" on page 347
"ReDim Statement" on page 377
"Static Statement" on page 460

UCase Function

This standard VB function returns a copy of a string after converting lowercase letters to uppercase.

Syntax UCase[$](*string*)

Argument	Description
string	A string or string expression

Returns A copy of *string* with lowercase letters replaced by uppercase letters.

Usage The translation is based on the country specified in the Windows Control Panel.

UCase accepts any type of argument and converts the input value to a string.

The dollar sign ($) in the function name is optional. If it is included, the return type is string. Otherwise the function typically returns a variant of vartype 8 (string). If the value of *string* is NULL, a variant of vartype 1 (Null) is returned.

Example This example converts a filename entered by a user to uppercase letters.

```
Option Base 1
Sub Button_Click
   Dim filename as String
   filename = "c:\temp\trace.txt"
   filename = UCase(filename)
End Sub
```

See Also "Asc Function" on page 88
"LCase Function" on page 289

UndoRecord Method

UndoRecord deletes an active record created by NewRecord in a Siebel business component. This method is used with business component objects. For details, read *Siebel Object Interfaces Reference*.

Unlock Statement

This standard VB statement controls access to an open file.

Syntax Unlock [#]*filenumber*[, { *record* | [*start*] To *end* }]

Argument	Description
filenumber	The file number used in the Open statement to open the file
record	An integer indicating the first record to unlock
start	A long integer indicating the first record or byte offset to unlock
end	A long integer indicating the last record or byte offset to unlock

Returns Not applicable

Usage For Binary mode, *start* and *end* are byte offsets. For random mode, *start* and *end* are record numbers. If *start* is specified without *end*, then only the record or byte at *start* is locked. If *end* is specified without *start*, then the records or bytes from record number or offset 1 to *end* are locked.

For input, output, and append modes, *start* and *end* are ignored and the whole file is locked.

Lock and Unlock always occur in pairs with identical parameters. Locks on open files must be removed before closing the file, or unpredictable results may occur.

Example For an example of the Unlock statement, read “Lock Statement” on page 300.

See Also
“Lock Statement” on page 300
“Open Statement” on page 344

Val Function

This standard VB function returns the numeric value of the first number found in a string.

Syntax Val(*string*)

Argument	Description
string	A string or string expression containing a number

Returns The value of the first number in *string*. If no number is found, Val returns 0.

Usage Spaces in the source string are ignored.

Example This example tests the value of the variable profit and displays 0 for profit if it is a negative number. The subroutine uses Sgn to determine whether profit is positive, negative, or zero.

```
Sub Button_Click
   Dim profit as Single
   Dim expenses
   Dim sales
   expenses = 100000
   sales = 20000
   profit = Val(sales)-Val(expenses)
   If Sgn(profit) = 1 then
      ‘Yeah! We turned a profit!
   ElseIf Sgn(profit) = 0 then
      ‘Okay. We broke even.
   Else
      ‘Uh, oh. We lost money.
   End If
End Sub
```

See Also

"CCur Function" on page 115
"CDbl Function" on page 117
"CInt Function" on page 123
"CLng Function" on page 128
"CSng Function" on page 137
"CStr Function" on page 139
"CVar Function" on page 143
"CVDate Function" on page 144
"Format Function" on page 202
"Str Function" on page 462

VarType Function

This standard VB function returns the variant type of the specified variant variable.

Syntax VarType(*varName*)

Argument	Description
varName	The name of the variant variable to query

Returns The value returned by VarType is one of the following:

Ordinal	Representation
0	(Empty)
1	Null
2	Integer
3	Long
4	Single
5	Double
6	Currency
7	Date
8	String
9	Object

Example This example returns the type of a variant.

```
Sub Button_Click
   Dim x
   Dim myarray(8)
   Dim retval
   Dim retstr
   myarray(1) = Null
   myarray(2) = 0
```

```
      myarray(3) = 39000
      myarray(4) = CSng(10^20)
      myarray(5) = 10^300
      myarray(6) = CCur(10.25)
      myarray(7) = Now
      myarray(8) = "Five"
      For x = 0 to 8
         retval = Vartype(myarray(x))
         Select Case retval
            Case 0
               retstr = " (Empty)"
            Case 1
               retstr = " (Null)"
            Case 2
               retstr = " (Integer)"
            Case 3
               retstr = " (Long)"
            Case 4
               retstr = " (Single)"
            Case 5
               retstr = " (Double)"
            Case 6
               retstr = " (Currency)"
            Case 7
               retstr = " (Date)"
            Case 8
               retstr = " (String)"
         End Select
         If retval = 1 then
            myarray(x) = "[null]"
         ElseIf retval = 0 then
            myarray(x) = "[empty]"
         End If
         Next x
   End Sub
```

See Also

"IsDate Function" on page 276
"IsEmpty Function" on page 277
"IsNull Function" on page 281
"IsNumeric Function" on page 283

WebApplet_InvokeMethod Event

The InvokeMethod event is triggered by a call to WebApplet.InvokeMethod or a specialized method, or by a user defined menu. For details, read *Siebel Object Interfaces Reference*.

Web_Applet_Load Event

The Load event is triggered when an applet is loaded. For details, read *Siebel Object Interfaces Reference*.

Web_Applet_PreCanInvoke Event

The PreCanInvokeMethod() event is called before the PreInvokeMethod, allowing the developer to determine whether or not the user has the authority to invoke a specified WebApplet method. For details, read *Siebel Object Interfaces Reference*.

WebApplet_PreInvokeMethod Event

The PreInvokeMethod event is called before a specialized method is invoked by the operating system, by a user defined applet menu, or by calling InvokeMethod on a web applet. For details, read *Siebel Object Interfaces Reference*.

WebApplet_ShowControl Event

This event allows scripts to modify the HTML generated by the Siebel Web Engine to render a control on a web page in a customer or partner application. For details, read *Siebel Object Interfaces Reference*.

WebApplet_ShowListColumn Event

This event allows scripts to modify the HTML generated by the Siebel Web Engine to render a list column on a web page. For details, read *Siebel Object Interfaces Reference*.

Weekday Function

This standard VB function returns the day of the week for a specified date-time value.

Syntax Weekday(*date*)

Argument	Description
date	An expression containing a date/time value

Returns An integer between 1 and 7, inclusive, representing a day of the week, where 1 = Sunday and 7 = Saturday.

Usage Weekday accepts any expression, including strings, and attempts to convert the input value to a date value.

The return value is a variant of vartype 2 (integer). If the value of *date* is NULL, a variant of vartype 1 (Null) is returned.

Example This example finds the day of the week on which November 7 occurs in the year 2009.

```
Sub Button_Click
   Dim checkdate
   Dim daynumber
   Dim msgtext
   Dim checkday as Variant
   Const checkyear = 2009
   Const checkmonth = 11
   Let checkday = 7
   checkdate = DateSerial(checkyear,checkmonth,checkday)
   daynumber = Weekday(checkdate)
   msgtext = "November 7, 2009 falls on a " & _
   Format(daynumber, "dddd")
End Sub
```

See Also

"Date Function" on page 146
"Date Statement" on page 147
"Day Function" on page 153
"Hour Function" on page 257
"Minute Function" on page 316
"Month Function" on page 320
"Now Function" on page 331
"Second Function" on page 397
"Year Function" on page 515

While...Wend Statement

This standard VB control structure controls a repetitive action.

Syntax

While *condition*

statement_block

Wend

Placeholder	Meaning
condition	A condition under which to execute the statements in ***statement_block***
statement_block	A series of statements to execute while *condition* is TRUE

Returns

Not applicable

Usage

The While statement is included in Siebel VB for compatibility with older versions of Basic. The Do...Loop statement is a more general and powerful flow control statement.

Example

This example opens a series of customer files and checks for the string `*Overdue*` in each file. It uses While...Wend to loop through the `c:\temP00?` files. These files are created by the subroutine `CreateFiles`.

```
(general) (declarations)
Option Explicit
Declare Sub CreateFiles

Sub CreateFiles
  Dim odue as String
  Dim ontime as String
  Dim x
  Open "c:\temp001" for OUTPUT as #1
  odue = "*Overdue*"
  ontime = "*On-Time*"
  For x = 1 to 3
     Write #1, odue
  Next x
```

```
  For x = 4 to 6
      Write #1, ontime
  Next x
  Close #1
  Open "c:\temp002" for Output as #1
  Write #1, odue
  Close #1
End Sub

Sub Button_Click
   Dim custfile as String
   Dim aline as String
   Dim pattern as String
   Dim count as Integer
   Call CreateFiles
   Chdir "c:\"
   custfile = Dir$("temP00?")
   pattern = "*" + "Overdue" + "*"
   While custfile <> ""
      Open custfile for input as #1
      On Error goto atEOF
      Do
          Line Input #1, aline
          If aline Like pattern Then
             count = count + 1
          End If
      Loop
nxtfile:
      On Error GoTo 0
      Close #1
      custfile = Dir$
   Wend
   Kill "c:\temp001"
   Kill "c:\temp002"
   Exit Sub
atEOF:
      Resume nxtfile
End Sub
```

See Also "Do...Loop Statement" on page 169

Width Statement

This standard VB statement sets the output line width for an open file.

Syntax Width [#]*filenumber*, *width*

Argument	Description
filenumber	The file number used in the Open statement to open the file
width	An integer expression indicating the width of the output line

Returns Not applicable

Usage A value of zero (0) for *width* indicates there is no line length limit. The default *width* for a file is zero (0).

Example This example puts five spaces and the string ABCD into a file. The five spaces are derived by taking 15 Mod 10, or the remainder of dividing 15 by 10.

```
Sub Button_Click
   Dim str1 as String
   Dim x as String * 10
   str1 = "ABCD"
   Open "C:\TEMP001" For Output As #1
   Width #1, 10
   Print #1, Spc(15); str1
   Close #1
   Open "c:\temp001" as #1 Len = 12
   Get #1, 1,x
   Close #1
   Kill "c:\temp001"
End Sub
```

See Also "Open Statement" on page 344
"Print Statement" on page 364

With Statement

This standard VB construct executes a series of statements on a specified variable.

Syntax With *variable*

statement_block

End With

Placeholder	Meaning
variable	The variable to be changed by the statements in *statement_block*
statement_block	The statements to execute on the variable

Returns Not applicable

Usage *Variable* can be an object or a user defined type. The With statements can be nested.

Example This example uses a Siebel VB method to change values in an object when a specific field is successfully changed. With is used to refer to the object in which the values are changed. For another example, read "Nothing Function" on page 329.

```
Sub BusComp_SetFieldValue(FieldName As String)

   Select Case FieldName
      Case "Account Status"
      If Me.GetFieldValue(FieldName) = "Inactive" Then
         Dim oBCact as BusComp
         Dim sMessage as String
         Set oBCact = me.BusObject.GetBusComp("Action")
         sMessage = "ADDED THRU SVB: Account Status made Inactive"

         With oBCact
            .NewRecord NewAfter
            .SetFieldValue "Type", "Event"
            .SetFieldValue "Description", sMessage
            .SetFieldValue "Done", _
               Format(Now(),"mm/dd/yyyy hh:mm:ss")
```

```
            .SetFieldValue "Status", "Done"
            .WriteRecord
         End With

         set oBCact = Nothing
      End If
      End Select

End Sub
```

See Also "Type Statement" on page 487

Write Statement

This standard VB statement writes data to an open sequential file.

Syntax

Write [#]*filenumber*[, *expressionList*]

Argument	Description
filenumber	The file number used in the Open statement to open the file
expressionList	One or more values to write to the file

Returns

Not applicable

Usage

The file must be opened in output or append mode. If *expressionList* is omitted, the Write statement writes a blank line to the file. For more information, read "Input Statement" on page 262.

NOTE: The Write statement results in quotes around the string that is written to the file.

Example

This example writes a variable to a disk file based on a comparison of its last saved time and the current time.

```
Sub Button_Click
   Dim tempfile
   Dim filetime, curtime
   Dim msgtext
   Dim acctno(100) as Single
   Dim x, I
   tempfile = "C:\TEMP001"
   Open tempfile For Output As #1
   filetime = FileDateTime(tempfile)
   x = 1
   I = 1
   acctno(x) = 0
   Do
      curtime = Time
      acctno(x) = 88
```

```
         If acctno(x) = 99 then
            If x = 1 then Exit Sub
            For I = 1 to x-1
               Write #1, acctno(I)
            Next I
            Exit Do
         ElseIf (Minute(filetime) + 2)< = Minute(curtime) then
            For I = I to x-1
               Write #1, acctno(I)
            Next I
         End If
         x = x + 1
      Loop
      Close #1
      x = 1
      msgtext = "Contents of C:\TEMP001 is:" & Chr(10)
      Open tempfile for Input as #1
      Do While Eof(1) <> -1
         Input #1, acctno(x)
         msgtext = msgtext & Chr(10) & acctno(x)
         x = x + 1
      Loop
      Close #1
         Kill "C:\TEMP001"
   End Sub
```

See Also

“Close Statement” on page 129
“Open Statement” on page 344
“Print Statement” on page 364
“Put Statement” on page 367

WriteRecord Method

WriteRecord commits to the database any changes made to the current record in a Siebel business component. For details, read *Siebel Object Interfaces Reference*.

Year Function

This standard VB function returns the year component of a date-time value.

Syntax

Year(*date*)

Argument	Description
date	An expression that can evaluate to a date/time value

Returns

An integer between 100 and 9999, inclusive.

Usage

Year accepts any type of *date*, including strings, and attempts to convert the input value to a date value.

The return value is a variant of vartype 2 (integer). If the value of *date* is NULL, a variant of vartype 1 (Null) is returned.

Example

This example returns the year for today.

```
Sub Button_Click
   Dim nowyear
   nowyear = Year(Now)
End Sub
```

See Also

"Date Function" on page 146
"Date Statement" on page 147
"Hour Function" on page 257
"Minute Function" on page 316
"Month Function" on page 320
"Now Function" on page 331
"Second Function" on page 397
"Time Function" on page 473
"WebApplet_InvokeMethod Event" on page 499

Siebel VB Compared to Other Basic Products A

This comparison covers the differences between Siebel VB and other Basic languages.

Differences Between Siebel VB and Earlier Versions of Basic

If you are familiar with versions of Basic that predate Windows, you may notice that Siebel VB includes many new features and changes from the language you have learned. Siebel VB more closely resembles other higher level languages popular today, such as C and Pascal.

The topics that follow describe some of the differences you may notice between the older versions of Basic and Siebel VB.

Line Numbers and Labels

Older versions of Basic require numbers at the beginning of every line. More recent versions do not support or require line numbers. Use of line numbers causes error messages.

Use a label to refer to a line of code. A label can be any combination of text and numbers. Usually it is a single word followed by a colon, placed at the beginning of a line of code. The Goto statement uses these labels.

Subroutines and Modularity of the Language

Because Siebel VB is a modular language, code is divided into subroutines and functions. The subprograms and functions you write use the Siebel VB statements and functions to perform actions.

Variable Scope

The placement of variable declarations determines their scope:

Scope	Definition
Local	Dimensioned inside a subprogram or function. The variable is accessible only to the subprogram or function that dimensioned it.
Module	Dimensioned in the (general) (declarations) section. The variable is accessible to any subprogram, function, or event attached to the object in whose Script window it appears.
Global	Dimensioned in the Application_Start event or Application.PreInvokeMethod method. The variable is accessible throughout the Siebel application. For more information, read Siebel Technical Note #217.

Data Types

Modern Basic is now a typed language. Siebel VB includes variants and objects, in addition to the standard data types of string, numeric, array, and record.

Variables that are defined as variants can store any type of data. For example, the same variable can hold integers or strings, depending on the procedure.

Objects give you the ability to manipulate complex data supplied by an application, such as windows, forms, or COM objects.

Financial Functions

Siebel VB includes a list of financial functions for calculating such things as loan payments, internal rates of return, or future values based on a company's cash flows.

Date and Time Functions

The date and time functions have been expanded to make it easier to compare a file's date to today's date, set the current date and time, time events, and perform scheduling-type functions.

Object Handling

Windows uses the Common Object Model (COM) standard for allowing supported applications to access one another's functionality. An object is the end product of a software application, such as a document from a word processing application. Therefore, the Object data type permits your Siebel VB code to access another software application through its objects and change those objects.

Environment Control

Siebel VB includes the ability to call another software application (AppActivate) and send the application keystrokes (SendKeys). Other environment control features include the ability to run an executable program (Shell), and return values in the operating system environment table (Environ).

Differences Between Siebel VB and Visual Basic

You may be familiar with any of several versions of Basic. The most common versions are Visual Basic and Visual Basic for Applications (VBA). Siebel VB shares a substantial common core of functions and statements with these versions, but each one has unique capabilities.

Siebel VB is very similar to Microsoft's Visual Basic, but there are some differences.

- "User Interface and Control-Based Objects" on page 521
- "Data Types" on page 521

User Interface and Control-Based Objects

Siebel VB does not provide for the inclusion of any Visual Basic user interface control objects, such as a Button Control. As a result, a VB property such as BorderStyle is not an intrinsic part of Siebel VB. Siebel VB allows you to reference the Siebel user interface controls and to set and retrieve their values. The Siebel user interface is managed with the Siebel Applet Designer. In keeping with this, the Visual Basic Input statement should not be used to acquire keyboard input.

Data Types

Siebel VB does not include a Boolean data type. You can simulate a Boolean data type by using an integer variable and regarding 1 (or any non-zero number) as TRUE and 0 as FALSE.

NOTE: If you need to call a field of DTYPE_BOOL in a script, you should declare it as a string.

Trappable Errors B

The following table (Table 9) lists the run-time errors that Siebel VB returns. These errors can be trapped by On Error. The Err function can be used to query the error code, and the Error function can be used to query the error text.

Table 9. Error Numbers and Strings

Error Code	Error Text
5	Illegal function call
6	Overflow
7	Out of memory
9	Subscript out of range
10	Duplicate definition
11	Division by zero
13	Type Mismatch
14	Out of string space
19	No Resume
20	Resume without error
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
52	Bad file name or number
53	File not found
54	Bad file mode

Table 9. Error Numbers and Strings

Error Code	Error Text
55	File already open
58	File already exists
61	Disk full
62	Input past end of file
63	Bad record number
64	Bad file name
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable set to Nothing
93	Invalid pattern
94	Illegal use of NULL
102	Command failed
429	Object creation failed
438	No such property or method
439	Argument type mismatch
440	Object error
901	Input buffer would be larger than 64K
902	Operating system error
903	External procedure not found
904	Global variable type mismatch

Table 9. Error Numbers and Strings

Error Code	Error Text
905	User-defined type mismatch
906	External procedure interface mismatch
907	Pushbutton required
908	Module has no MAIN
910	Dialog box not declared

Derived Trigonometric Functions C

Table 10 lists the trigonometric functions available in Siebel VB.

Table 10. Derived Trigonometric Functions

Function	Computed By
ArcCoSecant	ArcCoSec(x) = Atn(x/Sqr(x*x-1)) + (Sgn(x)-1)*1.5708
ArcCosine	ArcCos(x) = Atn(-x/Sqr(-x*x + 1)) + 1.5708
ArcCoTangent	ArcTan(x) = Atn(x) + 1.5708
ArcSecant	ArcSec(x) = Atn(x/Sqr(x*x-1)) + Sgn(x-1)*1.5708
ArcSine	ArcSin(x) = Atn(x/Sqr(-x*x + 1))
CoSecant	CoSec(x) = 1/Sin(x)
CoTangent	CoTan(x) = 1/Tan(x)
Hyperbolic ArcCoSecant	HArcCoSec(x) = Log((Sgn(x)*Sqr(x*x + 1) + 1)/x)
Hyperbolic ArcCosine	HArcCos(x) = Log(x + Sqr(x*x-1))
Hyperbolic ArcCoTangent	HArcCoTan(x) = Log((x + 1)/(x-1))/2
Hyperbolic ArcSecant	HArcSec(x) = Log((Sqr(-x*x + 1) + 1)/x)
Hyperbolic ArcSine	HArcSin(x) = Log(x + Sqr(x*x + 1))
Hyperbolic ArcTangent	HArcTan(x) = Log((1 + x)/(1-x))/2
Hyperbolic CoSecant	HCoSec(x) = 2/(Exp(x)-Exp(-x))
Hyperbolic Cosine	HCos(x) = (Exp(x) + Exp(-x))/2
Hyperbolic Cotangent	HCotan(x) = (Exp(x) + Exp(-x))/ (Exp(x)-Exp(-x))
Hyperbolic Secant	HSec(x) = 2/(Exp(x) + Exp(-x))
Hyperbolic Sine	HSin(x) = (Exp(x)-Exp(-x))/2

Table 10. Derived Trigonometric Functions

Function	Computed By
Hyperbolic Tangent	HTan(x) = (Exp(x)-Exp(-x))/(Exp(x) + Exp(-x))
Secant	Sec(x) = 1/Cos(x)

Glossary

call by reference
: Arguments passed by reference to a procedure can be modified by the procedure. Procedures written in Basic are defined to receive their arguments by reference. If you call such a procedure and pass it a variable, and if the procedure modifies its corresponding formal parameter, it modifies the variable. Passing an expression by reference is legal in Basic; if the called procedure modifies its corresponding parameter, a temporary value is modified, with no apparent effect on the caller.

call by value
: When an argument is passed by value to a procedure, the called procedure receives a copy of the argument. If the called procedure modifies its corresponding formal parameter, it has no effect on the caller. Procedures written in other languages, such as C, can receive their arguments by value.

comment
: A comment is text that documents a program. Comments have no effect on the program (except for metacommands). In Basic, a comment begins with a single quote and continues to the end of the line. If the first character in a comment is a dollar sign ($), the comment is interpreted as a metacommand. Lines beginning with the keyword Rem are also interpreted as comments.

control ID
: This can be either a text string, in which case it is the name of the control, or it can be a numeric ID. Note that control IDs are case-sensitive and do not include the dot that appears before the ID. Numeric IDs depend on the order in which dialog box controls are defined. You can find the numeric ID using the DlgControlID function.

function
: A procedure that returns a value. In Basic, the return value is specified by assigning a value to the name of the function, as if the function were a variable.

label
: A label identifies a position in the program at which to continue execution, usually as a result of executing a GoTo statement. To be recognized as a label, a name must begin in the first column and must be immediately followed by a colon (":"). Reserved words are not valid labels.

metacommand
: A metacommand is a command that gives the compiler instructions on how to build the program. In Basic, metacommands are specified in comments that begin with a dollar sign ($).

name A Basic name must start with a letter (A through Z). The remaining part of a name can also contain numeric digits (0 through 9) or an underscore character (_). A name cannot be more than 40 characters in length. Type characters are not considered part of a name.

precedence order Siebel VB's method to determine which operators in an expression to evaluate first, second, and so on. Operators with a higher precedence are evaluated before those with lower precedence. Operators with equal precedence are evaluated from left to right. The default precedence order (from high to low) is numeric, string, comparison, logical.

procedure A series of Siebel VB statements and functions executed as a unit. Both subprograms (Sub) and functions (Function) are called procedures.

subprogram A procedure that does not return a value.

vartype The internal tag used to identify the type of value currently assigned to a variant. This tag is one of the following:

```
Empty: 0
Null: 1
Integer: 2
Long: 3
Single: 4
Double: 5
Currency: 6
Date: 7
String: 8
Object: 9
```

Index

A

B

C

D

E

F

G

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y