

Siebel

VB Language Reference Guide

August 2026



Siebel
VB Language Reference Guide

August 2026

F87438-04

Copyright © 2020, Oracle and/or its affiliates.

Author: Sukanya Mishra

Contents

Get Help	i
1 What's New in This Release	1
What's New in Siebel VB Language Reference Guide, Siebel CRM 20.1 Update	1
What's New in Siebel VB Language Reference Guide, Siebel CRM 19.1 Update	1
2 About Siebel Visual Basic	3
About Siebel Visual Basic	3
Overview of Siebel Visual Basic	3
About Functions and Methods	4
Siebel VB and Other Versions of Visual Basic Programming Languages	5
3 Using Siebel VB	9
Using Siebel VB	9
About Data Types	12
About Expressions	18
About Object Handling	21
Declaring Procedures and Variables	22
Declaring Variables	24
About Formatting Strings	29
About Error Handling	37
4 Methods Reference for Siebel VB	45
Methods Reference for Siebel VB	45
Overview of Siebel VB Language Reference	45
Disk and Directory Control Methods	46
Accessing the File System in Script	51
File Control Methods	51
File Input and Output Methods	69
Code Setup and Control Methods	85

Code Control Statements	101
Variable Manipulation Methods	113
String Methods	125
Array Methods	146
Mathematical Methods	151
Date and Time Methods	165
ODBC Methods	180
Object Querying Methods	192
Financial Methods	198
Conversion Methods	204
COM Methods	209
Error Handling Methods	217

5 Quick Reference for Siebel VB Methods **225**

Quick Reference for Siebel VB Methods	225
Disk and Directory Control Quick Reference	225
File Control Quick Reference	226
File Input and Output Quick Reference	227
Code Setup and Control Quick Reference	228
Code Control Statements Quick Reference	229
Variable Manipulation Quick Reference	229
Strings Quick Reference	230
Arrays Quick Reference	232
Math Operations Quick Reference	232
Date and Time Quick Reference	233
ODBC Quick Reference	234
Object Querying Quick Reference	235
Financials Quick Reference	235
Conversions Quick Reference	236
COM Object Quick Reference	237
Error Handling Quick Reference	237

Get Help

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at <https://docs.oracle.com/>.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the [Oracle Accessibility Program website](#).

Contacting Oracle

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit [My Oracle Support](#) or visit [Accessible Oracle Support](#) if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an email to:
oracle_fusion_applications_help_ww_grp@oracle.com.

1 What's New in This Release

What's New in Siebel VB Language Reference Guide, Siebel CRM 20.1 Update

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

What's New in Siebel VB Language Reference Guide, Siebel CRM 19.1 Update

No new features have been added to this guide for this release. This guide has been updated to reflect only product name changes.

2 About Siebel Visual Basic

About Siebel Visual Basic

This chapter describes Oracle's Siebel Visual Basic. It includes the following topics:

- *Overview of Siebel Visual Basic*
- *About Functions and Methods*
- *Siebel VB and Other Versions of Visual Basic Programming Languages*

Overview of Siebel Visual Basic

Siebel VB (Visual Basic) is a programming language that provides the following capabilities:

- A fully functional procedural programming language
- An application interface that provides bidirectional access to Siebel business objects
- An editing environment that you can use to create and maintain custom Siebel VB code
- A debugger that you can use to help detect errors in your Siebel VB code
- A compiler that you can use to compile your custom Siebel VB code
- A run-time engine that is similar to a Visual Basic interpreter that you can use to process custom Siebel VB code

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

Siebel VB does not support the following items:

- Functionality developed through custom programming
- Automatic upgrades of custom code with the Siebel Application Upgrader
- Development of a separate, standalone application with Siebel VB
- Accessing server management functionality. To configure this functionality, you must use the user interface of the server management software or the command line.
- Development or deployment in a UNIX environment.

If you customize the Siebel Sales Enterprise application, then you must use caution. This customization must be done only by a trained technical professional.

Improper application configuration can adversely effect the reliability and performance of your Siebel application. You must thoroughly test any customization you develop before you implement your customization in a production environment.

Siebel VB and Unicode

Siebel VB supports Unicode except for functions that do the following:

- Perform a file input operation or a file output operation

- Accesses an external DLL that is dependent on character encoding and is not compliant with Unicode.

Typographic Conventions That This Book Uses

The following table describes the typographic conventions that this book uses.

Item That This Book Represents	Convention
Statements and functions.	Initial uppercase letters. For example: <code>Abs Len(variable)</code>
Arguments for statements or functions.	Letters that are in lowercase and that are italicized. A capital letter that occurs in a position other than the first letter can identify multiple English words. For example: • <code>variable</code> • <code>rate</code> • <code>prompt</code> • <code>stringVar</code>
Optional arguments or characters.	Square brackets. For example: • <code>[caption]</code> • <code>[type]</code> • <code>[\$]</code>
Required choice for an argument from a list of choices.	A list in curly brackets. An OR operator () separates the possible choices. For example: <code>{Goto label Resume Next Goto 0}</code>

For more information, see the following items:

- Creating script that runs on UNIX and that uses a Siebel object manager, see *Siebel Events Management Guide*.
- Creating, modifying, and deleting Siebel VB scripts that you use in Siebel Tools, see *Siebel Object Interfaces Reference*.

About Functions and Methods

A *Siebel VB function* is an independent section of code that does the following:

1. Receives information
2. Performs some action on this information
3. Returns a value to the item that called it

It begins with the following statement:

```
Function functionname
```

It ends with the following statement:

```
End Function
```

You can use the same format that you use with a variable to name a custom function. You can use any valid variable name as a function name. It is recommended that you use a name that describes the work that the function performs.

You can write code that calls a function repeatedly from various objects or script. It is similar to a subroutine. To call a function, you must know what information the function requires as input and what information it provides as output. This book describes the predefined functions that come with Siebel VB. You can use these functions any time you use the Siebel VB interpreter.

You can use a function anywhere you can use a variable. To use a function, you do the following:

- To declare it, you can use the function keyword.
- To determine the data that Siebel VB must pass to the function, you include the function operator. To include this operator, you can use a pair of parentheses immediately after the function name. For example, `TheApplication.RaiseErrorText()`.

A *Siebel VB method* is a function that is part of an object or class. It can include a predefined procedure that you can use to call a function.

A *Siebel VB statement* is a complete instruction.

For more information, see *Siebel eScript Language Reference* .

Siebel VB and Other Versions of Visual Basic Programming Languages

This topic compares Siebel VB to other versions of the Visual Basic programming language.

Differences Between Siebel VB and Earlier Versions of Visual Basic

Siebel VB is similar to a high level language, such as the C programming language or Pascal. This topic describes some of the differences you might notice between older versions of Visual Basic and Siebel VB.

Line Numbers and Labels

Older versions of Visual 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.

You can use a label to reference a line of code. A *label* can be any combination of text and numbers. Typically it is a single word followed by a colon, and placed at the beginning of a line of code. The Go To statement uses these labels.

Subroutines and Modularity

Because Siebel VB is a modular language, it divides code into subroutines and functions. To perform actions, the subroutines and functions you write use Siebel VB statements and functions.

How Declaring a Variable Affects Variable Scope

The following table describes how you declare a variable affects variable scope.

Scope	Where the Variable Is Declared
Local	Declared in a subroutine or function. Only the subroutine or function that declares the variable can access this local variable.
Module	Declared in the general declarations section. Any subroutine, function, or event that is attached to the object in the script window that displays this variable can access this modular variable.
Global	Declared in one of the following items: • Application_Start event • Application.PreInvokeMethod method You can write code that accesses a global variable throughout the Siebel application. For more information, see Siebel Technical Note #217 on My Oracle Support.

Data Types

Siebel VB is a typed language. It includes multiple data types, such as strings, numbers, variants, and arrays.

A variable that you define as a variant can store any type of data. For example, the same variable can hold integers or strings, depending on the code.

Objects allow you to manipulate complex data that an application supplies, such as Microsoft Windows forms, or COM objects.

For more information, see [About Data Types](#).

Financial Methods

Siebel VB includes financial methods that you can use to configure Siebel CRM to do a calculation. For example, to calculate a loan payment, an internal rate of return, or a future value according to a cash flow. For more information, see [Financial Methods](#).

Date and Time Methods

Date and time methods can compare a file date to the current date, set the current date and time, time events, and do scheduling. For more information, see [Date and Time Methods](#).

Methods to Access Other Applications

Microsoft Windows uses the Common Object Model (COM) standard to allow an application to access the functionality of another application. An object might be the end product of a software application. For example, a document from a word processing application. The Object data type allows Siebel VB to access another software application through these objects, and then modifies them. For more information, see [COM Methods](#).

Environment Control

Siebel VB can call another software application and send keystrokes to the application. It can also run code and return values in the operating system environment table.

Differences Between Siebel VB and Visual Basic

Siebel VB, Microsoft Visual Basic, and Visual Basic for Applications (VBA) use functions and statements that are similar to one another, but each of these languages possess some unique capabilities.

Boolean Data Type

Siebel VB does not include a Boolean data type. It considers 0 to be FALSE and any other numeric value to be TRUE. You can write code that uses only a numeric value as a Boolean value. A comparison expression always returns 0 for FALSE and -1 (negative one) for TRUE. You can use the following values in an integer variable to simulate a Boolean data type:

- **To represent TRUE.** A value of 1 or any number that is not zero.
- **To represent FALSE.** A value of 0.

If you must call a field from a script, and if this field is a DTYPE_BOOL type field, then you must declare it as a string.

User Interface and Control Objects

Siebel VB does not include any Visual Basic user interface control objects, such as a Button Control. A Visual Basic property such as BorderStyle is not part of Siebel VB. Siebel VB allows you to reference user interface controls in Siebel CRM and set and get their values. you can use Siebel Tools to manage a Siebel CRM user interface. You must not use the Input statement in Visual Basic as a way to get keyboard input.

3 Using Siebel VB

Using Siebel VB

This chapter describes how to use Siebel VB. It includes the following topics:

- *Guidelines for Using Siebel VB*
- *About Data Types*
- *About Expressions*
- *About Object Handling*
- *Declaring Procedures and Variables*
- *About Formatting Strings*
- *About Error Handling*

For information about the format that Siebel VB uses, see the topic that describes the format of the object interface method in *Siebel Object Interfaces Reference*.

Guidelines for Using Siebel VB

This topic describes guidelines that you can use when you program with Siebel VB. It includes the following topics:

- *Pass Values Through Reference*
- *Give Each Argument a Name*
- *Other Guidelines*

Pass Values Through Reference

Where possible, it is recommended that you pass a value through a reference and not through a variable. Passing a value *through a variable* is less efficient than passing the same value through a reference. You must write code that passes a value through a variable unless it cannot pass the same value through a reference.

Passing a Value Through a Reference

Siebel CRM can pass a variable to a subroutine or a function through a reference. Each method determines if it can receive a value from a variable or a reference. A subroutine or function can modify this value.

Passing a Value Through a Variable

Siebel CRM can pass a value to a function through a variable. After processing is complete, the variable retains the value that it contained before Siebel CRM passed it, even though the subroutine or function might modify the passed value.

If you configure Siebel CRM to pass a variable to a method that modifies the corresponding argument, and if you must retain the value that this variable contains, then you must enclose the variable in parentheses in the Call Subroutine statement. This format instructs Siebel VB to pass a copy of the variable. This technique is known as passing a *value through a variable*. For more information, see *Call Subroutine Method*.

If Siebel CRM passes a variable to a function, and if an argument for this function expects to receive a value through a reference, then this variable must match the exact type of the argument. This requirement does not apply to an expression or a variant.

If you configure Siebel CRM to call an external DLL, then you can configure the ByVal keyword to pass an argument through a value. You specify this configuration in the Declare Procedure statement or in the Call Subroutine statement. If you specify the ByVal keyword in the declaration, then the ByVal keyword is optional in the call. If you use the ByVal keyword, then it must precede the value. If you do not specify the ByVal keyword in the declaration, and if you specify the data type in the declaration, then you cannot use the ByVal keyword in the call.

For more information, see *Declare Procedure Method*.

Give Each Argument a Name

If you use a function that includes an argument, then you can provide a value for this argument. To do this, you list it in the order where it occurs in the format for the function. For example, assume you define the following function:

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

In this example, the myfunction function requires the following arguments:

- *id*
- *action*
- *value*

If you call this function, then you specify these arguments in the order that they occur. If a function includes multiple arguments, then it is recommended that you name each argument. This technique helps to make sure that Siebel CRM assigns each value that you specify to the correct argument.

If you give each argument a name, then you are not required to remember the order where the arguments occur. For example, the following format is correct even though the order varies from the order that the format specifies:

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

Consider the following code:

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

The following calls to this code 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
```

Format That You Can Use to Name an Argument

To name an argument, you use the following format:

```
argname:= argvalue
```

where:

- *argname* is the name of the argument that you specify in the Function statement or the Sub statement. For more information, see *Create Function Method* and *Create Subroutine Method*.
- *argvalue* is the value that Siebel CRM assigns to the argument when your code calls it.

For example:

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

Naming an Argument With More Complex Formats

With some formats, you must use a comma as a placeholder for each optional argument that you do not specify. If you name the arguments, then you can specify only the arguments that your code must use and their values. For example, consider the following code:

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

In this situation, you can use one of the following formats:

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

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

You cannot omit a required argument.

Where You Can Name an Argument

You can name an argument in the following situations:

- Functions you define with the Function statement
- Subroutines you define with the Sub statement
- Code that you declare with the Declare Procedure statement
- Some predefined functions and statements
- Some externally registered DLL functions and methods

Other Guidelines

It is recommended that you apply the following guidelines when you use Siebel VB. For information about each of them, see the topic about guidelines for using Siebel VB and Siebel eScript in *Siebel Object Interfaces Reference* :

- Declare your variables
- Use a standard naming convention
- Use a self-reference to identify the current object
- Avoid nested If statements

- Use a four-digit year
- Apply multiple object interface methods to a single object
- Write code that handles run-time errors

About Data Types

This topic describes the data types that Siebel VB uses. It includes the following topics:

- *Overview of Data Types*
- *Arrays*
- *Numeric Data Types That Siebel VB Uses*
- *Records*
- *Strings*
- *Variants*
- *Type Characters*
- *How Siebel VB Converts Data Types*
- *Comments*

Overview of Data Types

Siebel VB is a strongly typed language. A variable can contain data only of the declared type. It supports numeric, string, record, and array data that is standard with the Visual Basic programming language. It supports the following data types:

- Array
- Double-precision, floating-point number
- Double-precision integer
- Integer
- Object
- Record
- Single-precision, floating-point number
- String
- Variant

You can do one of the following to declare a variable:

- You can use a type character to implicitly declare a variable the first time your code references it. If you do not use a type character, then Siebel VB uses a default type of Variant.
- You can use the Declare Variable statement to explicitly declare the variable type. For more information, see *Declare Variable Statement*.

Arrays

To create an array, you specify one or more subscripts when you declare the array or when the Declare Array method redimensions the array. A subscript specifies the beginning and ending index for each dimension. If you specify only an ending index, then the beginning index depends on the Set Array Lower Boundary method. To reference an array element, you enclose each index value in parentheses after the array name. For example, the following format describes an array that includes three dimensions:

arrayName(i,j,k)

You can use an array with the following data types:

- Number
- String
- Variant
- Record
- Object

Siebel VB does not support the following items:

- An array of arrays
- Dialog box records
- Dialog box objects

For more information, see the following topics:

- *Declare Variable Statement*
- *Array Methods*
- *Declare Array Method*

For examples that use arrays, see the following topics:

- *Is Variable Set Method*
- *Is Variable Null Method*
- *Calculate Net Present Value Method*
- *Set Variant Variable to Null Method*
- *Set Array Lower Boundary Method*
- *Get Variant Type Method*

Dynamic Arrays

If you declare a dynamic array, then you do not specify a subscript range for the array elements. You can use the Declare Array method to set the subscript range. You can write code that sets the number of elements in a dynamic array according to other conditions that your code specifies. For example, you might use an array to store a set of values that the user enters, but you might not know in advance how many values the user will enter. In this situation, you can do one of the following:

- Dimension the array without specifying a subscript range, and then run a Declare Array method each time the user enters a new value.

- Write code that prompts the user to enter the number of values, and then run one Declare Array method to set the size of the array.

If you use the Declare Array method to modify the size of an array, and if you must preserve the contents of this array, then make sure you include the following Preserve argument in the Declare Array method:

Redim Preserve

```
ArrayName  
(  
  n  
)
```

If you use a Declare Variable statement to declare a dynamic array, then it can include no more than eight dimensions. You must use the Declare Array method to create a dynamic array that includes more than eight dimensions. This method allows you to declare an array that includes up to 60 dimensions. For more information, see [Declare Variable Statement](#) and [Declare Array Method](#).

You cannot use the Declare Array method to modify the number of dimensions of a dynamic array if the array already has dimensions. It can modify only the upper and lower boundaries of the dimensions of the array. For information about methods that can determine the current boundaries of an array, see [Get Array Lower Boundary Method](#) and [Get Array Upper Boundary Method](#).

Example of a Dynamic Array

The following example code uses a dynamic array named **varray** to hold the cash flow values that the user enters:

```
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
```

Numeric Data Types That Siebel VB Uses

The following table describes the numeric data types that Siebel VB uses.

Type	Description	Smallest Value	Largest Value
Integer	2 byte integer	Negative 32,768.	Positive 32,767.

Type	Description	Smallest Value	Largest Value
Long	4 byte integer	Negative 2,147,483,648.	Positive 2,147,483,647.
Single-Precision	4 byte floating-point number	Negative 3.402823e+38 0.0, 1.401298e-45.	Negative 1.401298e-45, 3.402823466e+38.
Double-Precision	8 byte floating-point number	Negative 1.797693134862315d+308, 0.0, 2.2250738585072014d-308.	Negative 4.94065645841247d-308, 1.797693134862315d+308.
Currency	8 byte number with a fixed decimal point	Negative 922,337,203,685,477.5808.	Positive 922,337,203,685,477.5807.

A numeric value is always signed.

You can write code that expresses an integer constant in the following ways:

- **Decimal.** You can use the decimal representation to express a decimal constant.
- **Octal.** You precede the constant with &O or with &o to express an octal value. For example, &o177.
- **Hexadecimal.** You precede the constant with &H or with &h to express a hexadecimal value. For example, &H8001.

For more information, see *Boolean Data Type*.

Records

A record is a data structure that includes one or more elements. Each of these elements includes a value. You must define a type first, and then declare the variable using that type. You must not include a type character as the suffix in the variable name. A record element uses dot notation. For example:

```
record.element
```

where:

- *record* is the record name.
- *element* is a member of this record. A record can contain elements that are themselves records.

Strings

A Siebel VB string can be one of the following:

- **Fixed.** You specify the length when you define the string. You cannot write code that modifies the length after it defines the string. A fixed string cannot be of 0 length.
- **Dynamic.** You do not specify a length. A dynamic string can vary in length from 0 to 32,767 characters.

There are no restrictions on the characters that a string can include. For example, a string can include a character whose ANSI value is 0.

You can cut and paste a character or you can use the Chr function to include a character from a character set. You can use characters only from the current character set. For more information, see [Get ANSI String Method](#).

If you configure Siebel CRM to exchange data with another application, then you must consider how this application handles terminating characters. Some applications create and expect only a carriage return. To stop output text, Siebel VB uses a carriage return and a line feed (CRLF). It expects CRLF characters in input text unless this input is specifically configured for some input functions.

Variants

To define a variable that contains any type of data, you can write code that uses the variant data type. To identify the type of data that the variable currently contains, Siebel VB stores a tag with the variant data. To examine this tag, you can use the VarType function.

The following table describes the types of values that a variant can contain.

Type	Size of Data	Smallest Value	Largest Value
0 Empty	0	Not applicable.	Not applicable.
1 Null	0	Not applicable.	Not applicable.
2 Integer	2 bytes, short	Negative 32768.	Positive 32767.
3 Long	4 bytes, long	Negative 2.147E9.	Positive 2.147E9.
4 Single	4 bytes, float	Negative 3.402E38.	Negative 1.401E-45.
		Positive 1.401E-45.	Positive 3.402E38.
5 Double	8 bytes, double	Negative 1.797E308.	Negative 4.94E-324.
		Positive 4.94E-324.	Positive 1.797E308.
6 Currency	8 bytes, fixed	Negative 9.223E14.	Positive 9.223E14.
7 Date	8 bytes, double	Jan 1, 100 to Dec 31, 9999.	Not applicable.
8 String	up to 2 gigabytes	Length is limited by the amount of random access memory, up to 2 gigabytes.	Not applicable.
9 Object	Not applicable.	Not applicable.	Not applicable.

Type	Size of Data	Smallest Value	Largest Value

If you define a variant that contains no data, then Siebel VB defaults the type to Empty. It does the following:

- Converts an empty variant to zero when it uses this variant in a numeric expression
- Converts an empty variant to an empty string when it uses this variant in a string expression

You can use the `IsEmpty` statement to determine if a variant is empty. For more information, see [Is Variable Set Method](#).

A null variant does not include data. It only represents a result that is not valid or that is ambiguous. You can use the `IsNull` statement to determine if a variant contains a null value. Null indicates that a variant is not set. For more information, see [Is Variable Null Method](#).

Type Characters

Siebel VB can use a special character as the suffix of the name of a function, variable, or constant. This character identifies the data type of the variable or function. It is a declaration.

The following table lists the characters you can use as a suffix.

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

How Siebel VB Converts Data Types

This topic describes the conversions that occur between the data types that Siebel VB supports. It does not support any other conversions. It does not automatically do conversions between numeric and string data:

- You can use the `Val` statement to convert a string to numeric data. For more information, see [Get First Number From String Method](#).
- You can use the `Str` statement to convert numeric data to a string. For more information, see [Convert Number to String Method](#).

Numeric Conversion

If Siebel VB converts data from a larger number type to a smaller number type, then a run-time numeric overflow might occur. This situation indicates that the value 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 float to a larger or a smaller type. Converting a long number to an integer is an example of converting a larger type to a smaller type.

String Conversion

If Siebel VB converts data from a fixed string to a dynamic string, then it creates a dynamic string that includes the same length and contents as the fixed string. If it converts a dynamic string to a fixed string, then it does the following work:

- If the dynamic string is shorter than the fixed string, then it extends the fixed string with spaces.
- If the dynamic string is longer than the fixed string, then it truncates the fixed string.

A string conversion does not cause run-time errors.

Variant Conversion

Siebel VB can convert data between any data type and a variant. It can convert a variant string to a number. If the variant string does not contain a valid representation of the number, then a type mismatch error occurs.

Comments

An apostrophe precedes a comment. It can occur on a separate line in the code or immediately after a statement or function on the same line. For example:

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

You can also use a Rem Statement to make a comment. For example:

```
Rem This is a comment line.
```

Siebel VB does not include a block comment feature.

About Expressions

An expression is a collection of two or more terms that perform a mathematical or logical operation. The terms are typically variables or functions that you use with an operator to evaluate to a string or numeric result. You can use an expression to perform a calculation, manipulate a variable, or concatenate a string.

Siebel VB evaluates an expression according to precedence order. You can use parentheses to override the default precedence order. The following operators are listed in order of highest precedence to lowest precedence:

1. *Numeric Operators*
2. *String Operators*
3. *Comparison Operators*
4. *Logical Operators*

Numeric Operators

The following table describes numeric operators.

Operator	Description
^ (caret)	Exponentiation.
- (minus) or + (plus)	Unary minus and plus.
* (asterisk) or / (forward slash)	Numeric multiplication or division. For division, the result is a Double.
\ (backward slash)	Integer division. The operands can be Integer or Long.
Mod	Modulus or Remainder. The operands can be Integer or Long.
- (minus) or + (plus)	Numeric addition and subtraction. You can also use the + (plus) operator for string concatenation.

String Operators

The following table describes string operators.

Operator	Description
& (ampersand)	String concatenation
+ (plus)	String concatenation

Comparison Operators

The following table describes comparison operators. For a number, Siebel VB increases the operands to the least common type:

- Integer is preferable to Long.
- Long is preferable to Single.
- Single is preferable to Double.

For a string, the comparison is case-sensitive and is according to the collating sequence that the language specifies in the Microsoft Windows Control Panel. The result is 0 for FALSE and negative 1 for TRUE.

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

Logical Operators

The following table describes logical operators. Siebel VB performs a bitwise operation for each operator.

Operator	Type	Description
NOT	Unary Not	Operand can be Integer or Long.
AND	And	Operands can be Integer or Long.
OR	Inclusive Or	Operands can be Integer or Long.
XOR	Exclusive Or	Operands can be Integer or Long.
EQV	Equivalence	Operands can be Integer or Long. (A EQV B) is the same as (NOT (A XOR B)).
IMP	Implication	Operands can be Integer or Long. (A IMP B) is the same as ((NOT A) OR B).

About Object Handling

An object is a reusable block of code. You can write code that instantiates an object or that does something. Each software application includes a set of properties and methods that modify the characteristics of an object.

A property affects how an object behaves. For example:

- Width is a property of a range of cells in a spreadsheet.
- Color is a property of a graph.
- Margin is a property of a word processing document.

A method causes an application to perform an action on an object. For example:

- Calculate for a spreadsheet
- Snap to Grid for a graph
- Auto-Save for a document

You can write Siebel VB that accesses a Siebel object and that modifies the properties and methods of this object. To access an object that is part of the Siebel application, you can run Siebel VB code that is external to the Siebel application.

To use a non-Siebel object in Siebel VB code, you must first assign it to an object variable. Assigning it instantiates it. To manipulate the object, you then reference the object name with or without properties and methods.

Example of Declaring an Object As a Siebel CRM Object Type

The following image includes an example that configures Siebel VB to access a Siebel object. You can declare an object as a Siebel CRM object type.

```
Sub BusComp_NewRecord
```

```
1 Dim oBC As BusComp  
  set oBC = me.GetPickListBusComp('Sales Stage')
```

```
2 oBC.ClearToQuery  
  oBC.ActivateField 'Sales Stage Order'  
  oBC.SetSortSpec 'Sales Stage Order'  
  oBC.ExecuteQuery ForwardOnly
```

```
3 set oBC = nothing
```

```
End Sub
```

Explanation of Callouts

To declare an object as a Siebel CRM object type, you do the following work:

1. You create an object variable to access the code. This example uses **as BusComp** to declare the object. It does not use **as Object**. This example instantiates the business component (BusComp) Siebel object type. You could declare it as an object, but if you use the methods associated with the object type, then you must declare it as the appropriate object type.
2. You can use methods and properties to manipulate the objects.
3. Set oBC to **nothing**. It is recommended that you always set an object to **nothing** when your code instantiates it.

You can use similar code to access other types of objects that are compliant with COM. You can use the software application that creates the object to modify properties and methods of the objects. For an example, see [Date and Time Methods](#).

Creating an Object Variable to Access an Object

The Declare Variable statement creates an object variable named **oBC** and assigns a picklist business component to this variable. The Assign COM Object statement uses a get method to assign the business component to the oBC variable. Note the following:

- If you instantiate an application, then you can use the GetObject method or the CreateObject method.
- If the application is already open on the Microsoft Windows desktop, then you use GetObject.
- If the application is not open, then you can use CreateObject.

For more information, see the following topics:

- [Declare Variable Statement](#)
- [Date and Time Methods](#)
- [Get COM Object Method](#)

Using Methods and Properties to Manipulate an Object

You can use the following format to access an object, property, or method:

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

For example, the GetPickListBusComp method of the BusComp object of the Siebel application is assigned to the oBC object variable. It returns the following value:

```
me.GetPickListBusComp("Sales Stage")
```

Declaring Procedures and Variables

This topic describes information about declaring procedures and variables.

Declaring a Procedure

This topic includes information about how to use the Declare Procedure statement to declare a procedure in a module or in a dynamic link library (DLL). For more information about this statement and the format and arguments that you can use with it, see [Declare Procedure Method](#).

Specifying the Data Type

You do one of the following to specify the data type for the value that a method returns:

- End the method name with a type character.
- Use the following clause:

As *funcType*

Note the following:

- If you do not specify a type, then the method that the Declare Procedure statement declares defaults to the data type variant.
- To use a record argument, you use an As clause and a *type* that is already defined with the Type statement.
- To use an array argument, you use empty parentheses after the *argument*. You do not specify an array dimension in the Declare Procedure statement.

Sequence Determines How You Must Declare Code

Siebel Tools compiles custom methods in alphabetical order. If you reference code in the current code before you define it, then you must use a declaration. For example, assume you create the following subroutines in the general declarations section:

```
Sub A
' Calling B
B
End Sub

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

In this situation, compilation fails with the following message:

Unknown function: B

If you add the following statement before Sub A, then the code compiles and runs properly:

```
Declare Sub B
```

Calling External DLL Code

You can use the Pascal calling convention to write code that calls external DLL code. Siebel VB pushes the arguments on the stack from left to right. It uses the Far reference to pass these arguments, by default. You can write code that uses the following keywords when it calls external DLL code:

- **ByVal**. Passes a value through a variable. Note the following:
 - You must specify ByVal before you specify the argument that it modifies.
 - If you apply ByVal to a numeric data type, then Siebel VB passes the argument through a variable, not through a reference.

- If you apply ByVal to a string data type, then Siebel VB passes the byFar pointer to the string data. It uses the byFar pointer to pass a string to a string descriptor, by default.

For more information, see *Pass Values Through Reference*.

- **Any.** Passes a value of any datatype. If you use Any for an argument, then Siebel VB does not examine the type of this argument. It does examine the type of any other argument that you do not specify as type Any. It uses the Far reference to pass the argument unless you specify the ByVal keyword. If you specify the ByVal keyword, then it does one of the following:
 - **Numeric data.** Places the value on the stack.
 - **String data.** Sets the pointer to the string.

The external DLL code must determine the type and size of the value.

If Siebel VB uses ByVal to pass a null string, then the external code receives a nonNULL character of 0. To send a NULL pointer, you must declare the argument as ByVal As Any, and then call the code with an argument of 0.

Declaring Variables

This topic includes information about using the Declare Variable statement to declare a variable. For more information about this statement and the format and arguments that you can use with it, see *Declare Variable Statement*.

It is recommended that you place procedure-level Declare Variable statements at the beginning of the procedure.

For information about explicitly declaring a variable, see *Force Explicit Declaration Statement*.

Determining Variable Scope

You can write code that shares a variable across modules. The following locations where you declare a variable determines the scope of the variable:

- **Declare in a procedure.** The variable is local to this procedure.
- **Declare outside a procedure.** The variable is local to the module.

If you declare a variable that has the same name as a module variable, then you cannot access the module variable. For more information, see *Declare Global Variable Statement*.

Specifying the Type When You Declare a Variable

You can specify one of the following types when you declare a variable:

- Arrays
- Numbers
- Records
- Strings
- Variants
- Objects

If you do not specify a data type, then Siebel VB assigns the variant data type to this variable.

If you do not include the *As* clause, then you can specify the *type argument*. To specify this argument, you use a type character as a suffix of the *variableName* argument. You can use both type specification techniques in a single *Declare Variable* statement. You cannot use them simultaneously on the same variable.

You can write code that omits the type character when your code references the variable. The type suffix is not part of the variable name.

For more information, see *About Data Types*.

Declaring an Array Variable

The following data types are available for an array:

- Numbers
- Strings
- Variants
- Records

You cannot write code that uses the *Declare Variable* statement to declare an array of arrays or an array of objects.

You include a subscript list as part of the *variableName argument* to declare an array variable. You can use one of the following formats:

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

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

The following table describes the *startSubscript* and *endSubscript* arguments.

Argument	Description
startSubscript	The index number of the first array element, followed by the following keyword: To
endSubscript	The index number of the last element of the array.

Argument	Description

Specifying Arguments When Declaring an Array

The `startSubscript` argument is optional. If you do not specify it, then Siebel VB uses zero as the default value. For example, the following statement creates an array named **counter** that includes elements 0 through 25, for a total of 26 elements. You can use the `Set Array Lower Boundary` statement to modify the default value:

```
Dim counter (25) as Integer
```

The values in the *startSubscript* argument and the *end Subscript argument* are valid subscripts for the array.

Size Limits of an Array

You can specify no more than 60 arrays in a parent array. The maximum total number of elements cannot exceed 65,536. For example, the following code is valid because 60 multiplied by 1092 is 65,520, which is less than 65,536:

```
Dim count(1 To 60, 1 To 1092)
```

The following code is not valid because 60 multiplied by 1093 is 65,580, which is more than 65,536:

```
Dim count(1 To 60, 1 To 1093)
```

Each subscript declares one array that resides in the parent array. If you do not specify the *subscriptRange argument*, then Siebel VB declares the array as a dynamic array. In this situation, you must use the `Declare Array` method to specify the dimensions of the array before your code can use it.

Declaring a Number Variable

Can use the `As` clause and one of the following numeric types to declare a numeric variable:

- Currency
- Integer
- Long
- Single
- Double

You can also include a type character as a suffix to the variable name to declare a numeric variable. Siebel VB sets a numeric variable to 0.

Declaring a Record Variable

You can use the `As` clause and specify a value in the *typeName argument* to declare a record variable. To define this type, you must use the `Type` statement before you can specify it in the *typeName argument*. You use the following format:

```
Dim  
variableName  
As  
typeName
```

A record includes a collection of data elements that are fields. Each field can be a numeric, string, variant, or previously defined record type. For more information on accessing fields in a record, see [Create Function Method](#).

Declaring a String Variable

Siebel VB supports the following types of strings:

- **Fixed-length.** Declared with a specific length between 1 and 32767. You cannot write code that modifies a fixed-length variable after you declare it. When you create a fixed-length string, Siebel VB fills it with zeros. To declare a fixed-length string, you use the following format:

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

- **Dynamic.** Does not include a declared length. It can vary in length from 0 to 32,767. The initial length for a dynamic string is 0. You can use one of the following formats to declare a dynamic string:

```
Dim  
variableName$  
Dim  
variableName  
As String
```

Declaring a Variant Variable

You declare a variable as a variant in the following situations:

- If the type of the variable is not known.
- If Siebel CRM might modify the variable type when the code runs. For example, a variant is useful for holding input from a user when valid input can include text or numbers.

You use one of the following formats to declare a variant variable:

```
Dim  
variableName  
Dim  
variableName  
As Variant
```

Siebel VB initializes a variant variable to the Empty variant type.

For more information, see *Variants*.

Declaring an Object Variable

To declare an object variable, you use the *As* clause and specify a class in the *typeName* argument. An object variable can reference an object. It can use dot notation to access members and methods of this object. For example:

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

You can declare an object as *New* for some classes. For example:

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

A *Set* statement is not required in this situation. Siebel VB allocates a new object when it uses this variable.

You cannot use the *New* operator with the Basic Object class.

Caution About Declaring Multiple Variables on One Line

CAUTION: You can declare multiple variables on one line. However, if you do not include the type for each variable, then Siebel VB applies the type of the last variable to all the variables that you declare on this line.

For example, the following code declares all of the following variables as strings:

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

Shared Keyword Allows Backward Compatibility

Siebel VB includes the *shared* keyword to support backward compatibility with older versions of Visual Basic. You cannot use it in a *Declare Variable* statement in a procedure. If you use the *Shared* keyword in a *Declare Variable* statement in a procedure, then it has no effect.

About Formatting Strings

This topic includes information about how to use the Set String Format method to format an output string. It includes the following topics:

- *Numeric Formats*
- *Date and Time Formats*
- *Other Formatting Options*

For more information, see *Set String Format Method*.

Numeric Formats

This topic describes numeric formats that you can use with the Set String Format method.

Predefined Numeric Formats

The following table describes the predefined numeric formats that you can use.

Format	Description
General Number	Displays the number without a 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 a 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. Uses a thousand separator and displays two digits to the right of decimal separator. Encloses negative value in parentheses.
Percent	Multiplies the number by 100 and displays it with a percentage symbol (%) appended to the right. Displays two digits to the right of the decimal separator.
True or False	Displays FALSE for 0, or TRUE for any other number.
Yes or No	Displays No for 0, or Yes for any other number.
On or Off	Displays Off for 0, or On for any other number.

Custom Numeric Formats

You can use one or more digit characters to create a simple custom numeric format. You can use the following digit characters:

- **0 (zero).** Displays a corresponding digit in the output.
- **(#) number sign.** If the digit is significant, then it displays it in the output. A significant digit is a digit that resides in the middle of the number or is not zero.

The following table includes examples of using zero and the number (#) sign.

Number	Format	Result
1234.56	#	1235
1234.56	###	1234.56
1234.56	##	1234.6
1234.56	#####.##	1234.56
1234.56	00000.000	01234.560
0.12345	###	.12
0.12345	0.##	0.12

You can use a decimal separator as an option.

A comma instructs Siebel VB to place a comma between every three digits that occur to the left of the decimal separator.

The following table includes examples of using a comma.

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

Siebel VB uses the current international settings for your computer to determine the character to display for a comma or a period. For example, some locales use a period as the decimal separator. Other locales use a comma.

Scaling Numbers

You can do one of the following to scale a number:

- Insert one or more commas before the decimal separator. Each comma that precedes the decimal separator divides the number by 1000. If you do not specify a decimal separator, then this configuration applies to all digits. Siebel CRM does not include a comma in the output string.
- Include a percentage symbol (%) in the format argument. A percentage symbol multiplies the number by 100. Siebel CRM includes the percentage symbol in the output string in the same position where it occurs in the *format argument*.

The following table includes examples of using a comma or a percentage symbol to scale numbers.

Number	Format	Result
1234567.8901	#,##	1234.57
1234567.8901	#,,####	1.2346
1234567.8901	#,#,##	1,234.57
0.1234	#0.00%	12.34%

Inserting Characters In Number Formats

To insert a character in a number in the output string, you enclose the character in double quotes in the format argument. You can also insert a set of characters. Siebel VB inserts the following characters in the output string in a location that matches the position in the format argument:

- + \$ (space

You can precede the character with a backslash (\) to insert a single character.

The following table includes examples of using double quotes and backslashes to insert characters.

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 Get ANSI String method to insert a quotation mark (") in a format argument. The character code for a quotation mark is 34. For more information, see [Get ANSI String Method](#).

Scientific Notation Formats

You can include one of the following exponent strings in the *format* argument to format a number in scientific notation:

- E-
- E +
- e-
- e +

Siebel VB displays this notation in the following ways:

- **An uppercase e.** An uppercase e displays in the output.
- **A lowercase e.** A lowercase e displays in the output.
- **A minus sign that follows an uppercase e.** A minus sign precedes any negative exponent that displays in the output.
- **A plus sign.** A sign always precedes the exponent in the output.

You precede the exponent string with one or more digit characters. The number of digit characters that following the exponent string determines the number of exponent digits that occur in the output.

The following table includes examples of using exponential notation.

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

Using Sections In a Numeric Format

A numeric format can include up to four sections. A semicolon (;) separates each section. The format varies depending on the number of sections you specify:

- **One section.** This section applies to every value.
- **Two sections:**
 - The first section applies to positive values and zeros.
 - The second section applies to negative values.

- **Three sections:**

- The first section applies to positive values.
- The second section applies to negative values.
- The third section applies to zeros.

If you include semicolons with nothing between them, then Siebel VB uses the format of the first section to print the undefined section.

- **Four sections.** Same as three sections, except the fourth section applies to Null values. If you do not include the fourth section, and if the input expression results in a NULL value, then Siebel VB returns an empty string.

The following table includes examples of using sections.

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

Date and Time Formats

This topic describes date and time formats that you can use with the Set String Format method. For more information, see [Set String Format Method](#).

Predefined Date and Time Formats

The following table describes predefined date and time formats that you can use.

Format	Description
General Date	Note the following: • If the number includes an integer part and a fractional part, then it displays date and time information. For example, 11/8/2011 1:23:45 PM. • If the number includes only an integer part, then it displays this integer as a date. • If the number includes only a fractional part, then it displays this fractional part as time.
Long Date	Displays a long date. The International section of the Control Panel defines a long date.

Format	Description
Medium Date	Displays the date using the month abbreviation without the day of the week. For example, 08-Nov-2011.
Short Date	Displays a short date. The International section of the Control Panel defines a short date.
Long Time	Displays a long time. The International section of the Control Panel defines a long time. It includes hours, minutes, and seconds.
Medium Time	Does not display seconds. It displays hours in a 12 hour format and uses the AM and PM designator.
Short Time	Does not display seconds. It uses a 24 hour format and does not use the AM and PM designator.

Custom Date Formats

You can use a series of tokens in the *format* argument to define a custom format for a date. Siebel VB replaces each token in the output string with an appropriate corresponding value.

The following table describes the tokens that you can use.

Token	Output
c	The equivalent of the format <i>dddd tttt</i> .
dddd	The current date, including the day, month, and year according to the current Short Date setting of the computer. The following format is the default Short Date setting for the United States: <i>m/d/yy</i>
dddddd	The current date, including the day, month, and year according to the current Long Date setting of the computer. The following format is the default Long Date setting for the United States: <i>mmm dd, yyyy</i>
tttt	The current time, including the hour, minute, and second using the current time settings of the computer. The following format is the default time setting for the United States: <i>h:mm:ss AM/PM</i>

Specifying Individual Parts of a Custom Date Format

The following table describes the tokens that you can use to specify individual parts of a custom date format.

Token	Output
d	The day of the month as a one or two digit number in the range of 1 through 31.

Token	Output
dd	The day of the month as a two digit number in the range of 1 through 31.
ddd	The day of the week as a three letter abbreviation in the range of Sun through Sat.
dddd	The day of the week without abbreviation in the range of Sunday through Saturday.
w	The day of the week as a number, where Sunday is 1 and Saturday is 7.
ww	The week of the year as a number in the range of 1 through 53, where the first week of January is always week 1.
m	The month of the year or the minute of the hour as a one or two digit number: • If the preceding token is an hour, then the minute is the output. • If the preceding token is not an hour, then the month is output.
mm	The month or the year or the minute of the hour as a two digit number: • If the preceding token is an hour, then the minute is the output. • If the preceding token is not an hour, then the month is output.
mmm	The month of the year as a three letter abbreviation in the range of Jan through Dec.
mmmm	The month of the year without abbreviation in the range of January through December.
q	The quarter of the year as a number in the range of 1 through 4.
y	The day of the year as a number in the range of 1 through 366.
yy	The year as a two digit number in the range of 00 through 99.
yyyy	The year as a three digit or a four digit number in the range of 100 through 9999.
h	The hour as a one digit or a two digit number in the range of 0 through 23.
hh	The hour as a two digit number in the range of 00 through 23.
n	The minute as a one digit or a two digit number in the range of 0 through 59.
nn	The minute as a two digit number in the range of 00 through 59.
s	The second as a one digit or a two digit number in the range of 0 through 59.

Token	Output
ss	The second as a two digit number in the range of 00 through 59.

Using a 12 Hour Format

The following table describes the tokens that you can use that use a 12 hour format. Siebel VB uses a 24 hour format, by default.

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

Other Formatting Options

This topic describes other formatting options that you can use with the Set String Format method.

Changing Formatting Sequence

Siebel VB formats characters from left to right, by default. To format characters from right to left, you can include an exclamation point (!) in the format argument.

Changing Case

Siebel VB does not modify the case of characters that it formats, by default. To instruct Siebel VB to modify the case of a character, you can use the following characters:

- < (Less than). Converts output characters to lowercase.
- > (Greater than). Converts output characters to uppercase.

Handling Spaces That Occur in the Input String

The following table describes characters that you can use to handle spaces that occur in the input string.

Character	Description
@	Siebel VB displays a character or a space according to the following logic: • If a character resides in the string in the position where the @ occurs in the format string, then it displays this character in this position • If no character resides in the string in the position where the @ occurs in the format string, then it displays a space in this position
&	Siebel VB displays a character or nothing according to the following logic: • If a character resides in the string in the position where the & occurs in the format string, then it displays this character in this position • If no character resides in the string in the position where the & occurs in the format string, then it displays nothing

About Error Handling

This topic describes error handling. It includes the following topics:

- [Overview of Error Handling](#)
- [Handling Errors That Siebel VB Returns](#)
- [Handling Custom Errors](#)
- [Handling Errors That a Siebel VB Method Returns](#)
- [Error Code and Error Text for Siebel VB Errors](#)

Overview of Error Handling

Siebel VB includes the following error handling statements and functions:

- Err
- Error
- On Error

Siebel VB returns a code for many of the run-time errors that you might encounter. For more information, see [Error Code and Error Text for Siebel VB Errors](#).

You can write code that uses the On Error statement in the following ways:

- Add code that handles the error immediately before a line of code where an error might occur. For example, after a File Open statement.
- Label a separate section of the code only for error handling and instruct Siebel VB to proceed to this label if an error occurs.

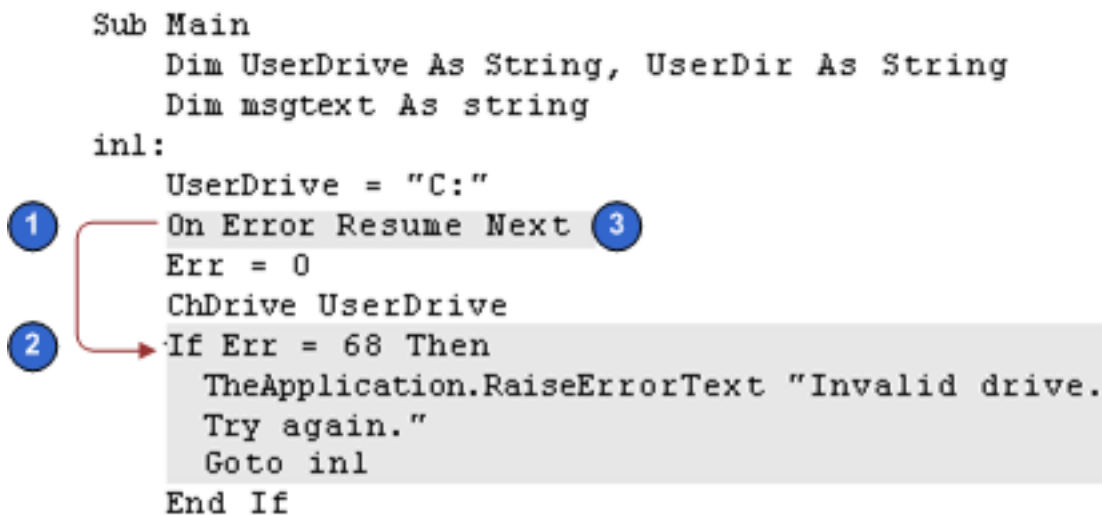
Handling Errors That Siebel VB Returns

This topic describe how to write code that handles the errors that Siebel VB returns.

Using the Body of the Code to Handle Siebel VB Errors

To handle errors in the body of code, you place the code that handles the error immediately before the line of code that could cause an error.

The following image includes an example that handles errors in the body of the code.



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

The image shows a screenshot of Siebel VB code with three callouts. Callout 1 points to the 'On Error Resume Next' statement. Callout 2 points to the 'If Err = 68 Then' statement. Callout 3 points to the 'Resume Next' argument in the 'On Error' statement.

Explanation of Callouts

The example that handles errors in the body of the code includes the following items:

1. The On Error statement identifies the line of code to run if an error occurs.
2. The If statement handles the error. It uses the Err statement to identify the error that Siebel VB returns.
3. The Resume Next argument instructs Siebel VB to proceed to the next line of code after it handles the error.

Using an Error Handler to Handle Siebel VB Errors

The following image includes an example that uses an error handler to handle errors.



```
1 On Error Goto Errortrap1
in2:
  UserDir = "test"
  ChDir UserDrive & "\" & UserDir
2 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
```

Explanation of Callouts

The example that uses an error handler to handle errors includes the following items:

1. The On Error statement identifies the line of code that Siebel CRM runs if an error occurs. The code segment is part of the main code and it uses the Err statement to determine the error code that Siebel VB returns.
2. You precede the code with an Exit statement to make sure that it does not accidentally proceed to the error handler.

Handling Custom Errors

You can create a custom set of error codes to handle errors that are specific to your code. For example, you can create your own set of error codes if your Siebel VB code creates rules for file input but the user does not follow these rules. You can configure Siebel VB to create an error and reply appropriately using the same statements and functions that you use for error codes that Siebel VB returns.

Using the Body of the Code to Handle Custom Errors

The following image includes an example that uses the body of the code to handle a custom error.

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

Explanation of Callouts

The example that uses the body of the code to handle a custom error includes the following items:

1. Place the code that handles the error immediately before the line of code that could cause an error.
2. You use the Error statement to set the custom error to a value of 30000.

Using a Label to Handle Custom Errors

The following image includes an example that uses a label to handle a custom error.

```
Sub Main
  ❶ 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:
  ❷ 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
```

Explanation of Callouts

The example that uses a label to handle a custom error includes the following items:

1. Place the code that handles the error immediately before the line of code that could cause an error.
2. Use a labeled section of code to handle the custom error.

Handling Errors That a Siebel VB Method Returns

You must configure Siebel CRM to handle an error that a Siebel VB method returns differently from how you configure it to handle an error that a Visual Basic function or statement returns. You can use the following code to handle an error that a Siebel VB method creates. This code displays the text of the error message:

DisplayError:

```
If ErrCode <> 0 Then

  ErrText = GetLastErrText

  TheApplication.RaiseErrorText ErrText
Exit Sub
End If
```

Note the following:

- A Siebel VB method uses numeric error codes in the range of 4000 through 4999.
- `DisplayError` is a label and is the target of a `Go To` statement that exists elsewhere in the code.
- The `GetLastErrorText` method is available only through an interface that is external to Siebel Tools. You can use it in Microsoft Visual Basic but not in Siebel VB.

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

Error Code and Error Text for Siebel VB Errors

The following table lists the run-time errors that Siebel VB returns. The `On Error` statement can handle these errors. The `Err` function can query the error code and the `Error` function can query the error text. For more information, see the following topics:

- [Open File Method](#)
- [Get Error Code Method](#)
- [Get Error Code Line Method](#)

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

Error Code	Error Text
48	Error in loading DLL
52	Bad file name or number
53	File not found
54	Bad file mode
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

Error Code	Error Text
438	No such property or method
439	Argument type mismatch
440	Object error
901	Input buffer is larger than 64K
902	Operating system error
903	External procedure not found
904	Global variable type mismatch
905	User-defined type mismatch
906	External procedure interface mismatch
907	Pushbutton required
908	Module has no MAIN
910	Dialog box not declared

4 Methods Reference for Siebel VB

Methods Reference for Siebel VB

This chapter describes reference information for Siebel VB methods. It includes the following topics:

- *Overview of Siebel VB Language Reference*
- *Disk and Directory Control Methods*
- *File Control Methods*
- *File Input and Output Methods*
- *Code Setup and Control Methods*
- *Code Control Statements*
- *Variable Manipulation Methods*
- *String Methods*
- *Array Methods*
- *Mathematical Methods*
- *Date and Time Methods*
- *ODBC Methods*
- *Object Querying Methods*
- *Financial Methods*
- *Conversion Methods*
- *COM Methods*
- *Error Handling Methods*

Overview of Siebel VB Language Reference

A Siebel VB method can access a component of the Siebel software architecture, such as applets and business components. You must preface a Siebel VB method with the name of the architecture component that it references. For example:

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

where:

- BusComp is the name of the architecture component
- GetFieldValue is the name of the Siebel VB method

A Microsoft VB command does not reference a specific component of the Siebel software architecture. All the statements and methods that this chapter describes are Microsoft VB constructs except for the ODBC methods. For more information, see *ODBC Methods*.

For more information, see *About Functions and Methods*.

Usage of the Dollar Sign

Some methods include the dollar sign (\$) in the method name. This dollar sign is optional:

- If you include it, then the return type is string.
- If you do not include it, then the return type is string variant.

This situation is true unless noted differently in the description for each method in this chapter.

For more information, see *Variants*.

Methods, Functions, and Statements Described in Siebel Object Interfaces Reference

Siebel Object Interfaces Reference describes a number of methods, functions, and statements that you can use with Siebel VB that this chapter does not describe. For more information about each of these methods, see the Siebel VB Quick Reference chapter in *Siebel Object Interfaces Reference*.

Disk and Directory Control Methods

This topic describes methods that you can use to control the disk and directories. It includes the following topics:

- *Change Directory Method*
- *Change Drive Method*
- *Create Directory Method*
- *Get Current Directory Method*
- *Remove Directory Method*

Change Directory Method

The Change Directory method changes the default directory of a drive. It does not return a value. It does the following depending on how you use the arguments:

- **Include the drive argument.** It changes the default directory on the current drive.
- **Include the first backslash in [\\]directory\.** It uses the path from the root directory.
- **Do not include the first backslash in [\\]directory\.** It changes to a directory that resides in the current directory.

The Change Directory method does not change the default drive. You can use the Change Drive method to change the default drive.

Format

```
ChDir [drive][[\\]directory\\]directory
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>drive</i>	The name of the drive that contains the desired default directory. You can use one of the following values: • A letter • A string expression that identifies the drive name A colon is not required.
[\\] <i>directory</i> \\	The path. You can use one of the following values: • Path to the directory that this method sets as the default directory. • A string expression that identifies the path. You can use this argument in the following situations: • The directory is not in the current directory of the specified drive • You do not specify a drive and the directory is not in the default drive
<i>directory</i>	The name of the directory that this method sets for the default directory, or a string expression that identifies this name.

Example

The following example changes the current directory to **C:\Windows**:

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

Change Drive Method

The Change Drive method changes the default drive. It does not return a value. The drive that it changes to the default drive must exist, and this drive must reside in the range that the LASTDRIVE statement in the config.sys file specifies.

You can use a colon as part of the name of the drive but it is not required.

You can use the Change Directory method to change the current directory on a drive.

Format

ChDrive *drive*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>drive</i>	A string expression that identifies the drive that this method makes the default drive. This method changes the drive according to one of the following values that you provide in the drive argument: - Null string (""). The default drive remains the same. - A string. It uses the first letter only. - You do not include the drive argument. It displays an error message.

Example

The following 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
```

Create Directory Method

The Create Directory method creates a new directory. It does not return a value.

Format

MkDir [*drive*:] [*\directory*] *directory*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>drive</i>	Optional. The name of the drive where this method creates the directory. You can use a letter or a string expression that identifies the drive name. If you do not include the <i>drive argument</i> , then this method creates the new directory on the current drive. If you include the <i>drive argument</i> , then you must include the colon (:).
<i>\directory\</i>	The path to the directory where this method creates the new directory, or a string expression that identifies this path.

Argument	Description
	You can use this argument only if the Create Directory method must not create the directory in the following locations: - On the current directory of the drive that the drive argument identifies - On the default drive if you do not include the drive argument
<i>directory</i>	The name of the directory that this method creates, or a string expression that identifies this directory name.

Example

The following example creates a new temporary directory in the **C:** directory, 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
```

Get Current Directory Method

The Get Current Directory method returns the default directory of a drive. If you specify a null ("") drive argument, or if you do not include the drive argument, then it returns the path for the default drive.

The drive that this method examines must exist, and it must reside in the range that the LASTDRIVE statement in the config.sys file specifies.

You can use a colon as part of the name of the drive but it is not required.

You can use the Change Drive method to change the current drive. You can use the Change Directory method to change the current directory.

Format

```
CurDir[$] [(drive)]
```

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>drive</i>	The letter of the drive to search.

Example

The following example changes the current directory to **C:\Windows**:

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

Remove Directory Method

The Remove Directory method removes a directory. It does not return a value. Note the following:

- The directory that it removes must be empty, except for the working directory or parent directory.
- It cannot remove the default directory. To remove the default directory, you must first make another directory current on the drive.

Format

Rmdir [*drive:*] [*\directory*] *directory*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>drive</i>	Optional. The name of the drive that contains the directory that this method removes. You can use a letter or a string expression that identifies the drive name.
<i>\directory\</i>	The path to the directory that this method removes. You can use this argument only if the Remove Directory method must not remove the directory from the following locations: • On the current directory of the drive that the drive argument identifies • On the default drive if you do not include the drive argument
<i>directory</i>	The name of the directory that this method removes.

Example

The following example makes a new temporary directory in the **C:** directory, 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
```

Accessing the File System in Script

Various methods allow your script to interact with or run commands on the operating system, file system, and other software running on the Siebel Server. You must explicitly allow this by setting System Preferences. This does not mean that the Siebel script is allowed/disallowed executing, it means that certain statements within those script can be denied access to the operating system. The script will still run, but, if access to the operating system is disallowed, a runtime error will occur when the lines that attempt to access the operating system are encountered at runtime.

Note: These two system preferences can be set and used independently.

Runtime Scripts System Access (default is false)

A Runtime Script is the script that you write in the Administration - Business Service Screen. Since these Business Service Methods are not compiled, they are less secure. Not all client-side Business Services need to access the file system to run commands, but if yours do, you must explicitly allow them access to the file system by adding this System Preference and setting its value to **true**.

Compiled Scripts System Access (default is true)

The Business Service Script that you write in Web Tools or Siebel Tools is compiled. By default, these methods are allowed to access the file system and run commands such as *GetObject*. If you want to disallow any compiled Business Service Script from accessing the file system and running commands on it you must add this System Preference and set its value to **false**.

Some examples (not exhaustive) of methods that access the operating system:

- *GetObject*
- *Open/Close/Kill/Dir* and all File methods
- *Clipboard*
- *SendKeys*

File Control Methods

This topic describes methods that you can use to control files. It includes the following topics:

- *Close All Files Method*
- *Close File Method*
- *Copy File Method*
- *Delete File Method*

- *Get File Attributes Method*
- *Get File Date Method*
- *Get File Length Method*
- *Get File Length 2 Method*
- *Get File Mode Method*
- *Get File Names Method*
- *Get Free File Number Method*
- *Lock File Method*
- *Open File Method*
- *Rename File Method*
- *Set File Attributes Method*
- *Unlock File Method*

Close All Files Method

The Close All Files method closes every open file and writes to disk any data that currently resides in the operating system buffers. It does not return a value.

Format

Reset

This method does not include any arguments.

Example

The following example creates a file, puts the numbers 1 through 10 in this file, and then attempts to get past the end of the file. The On Error statement handles the error and Siebel VB continues to the Debugger code. This code uses the Reset statement to close the file before it exits:

```
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
```

Close File Method

The Close File method closes a file and stops any input or output operations to this file. It does not return a value. When it runs, Siebel VB writes the final output buffer to the operating system buffer for this file. It frees the buffer space that is associated with the closed file. You can use the Close All Files method to cause the operating system to move the data in the buffers to disk. For more information, see [Close All Files Method](#).

Format

```
Close [[#]filename [, [#]filename ... ]]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file. It identifies the file to close. <i>The value in the filename argument</i> is the number that the Open statement assigns to the file. A pound sign (#) can precede this argument. If you do not include this argument, then the Close statement closes every open file. When a Close statement runs, the association of a file with <i>filename</i> is ended, and Siebel VB can reopen this file with the same or a different file number.

Example

The following example opens a file for random access, gets the contents of one variable, and then closes the file. The CreateFile subroutine creates the **c:\temp001** file that the main subroutine uses:

```
(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
```

Copy File Method

The Copy File method copies a file. It does not return a value. You cannot use the following wildcards in any of the arguments:

- * (asterisk)
- ? (question mark)

The Copy File method *cannot copy the file that the source argument identifies* in the following situations:

- If Siebel VB opens this file for anything other than read access
- If other code has already opened and is using the file

Format

```
FileCopy [path1] source, [path2] target
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>path1</i>	The path of the file to copy. If <i>the value in the source</i> argument does not reside in the current directory, then this argument is optional.
<i>source</i>	The name and, if necessary, the path of the file to copy.
<i>path2</i>	The path to a directory. This method copies the file to this directory. If you require that this method not copy the file to the current directory, then the path2 argument is optional.
<i>target</i>	A name. This method copies the file to this name.

Example

The following example copies one file to another file:

```
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
```

Delete File Method

The Delete File method deletes a file. It does not return a value. It only deletes files. It does not delete directories. You can use the Delete Directory method to delete a directory.

Format

```
Kill  
  pathname
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string expression that identifies a valid DOS file specification. It can include paths and the following wildcards: • * (asterisk) • ? (question mark)

Example

The following example does the following work:

1. Prompts a user for an account number.
2. Opens a file.
3. Searches the file for the account number.
4. Displays the matching letter for that number.

The **CreateFile** subroutine creates the **c:\temp001** file that the main subroutine uses. The first subroutine uses the Kill statement to delete the file after Siebel CRM finishes processing:

```
(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
```

Get File Attributes Method

The Get File Attributes method returns the attributes of a file, directory, or a volume label.

Format

GetAttr(*pathname*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string or string expression that evaluates to the name of the file, directory, or volume label to query. It cannot include the following wildcards: * (asterisk) ? (question mark)

Returns

The following table describes the values that the Get File Attributes method returns.

Value	Description
0	Normal file.
1	Read-only file.
2	Hidden file.
4	System file.

Value	Description
8	Volume label.
16	Directory.
32	Archive. The file has changed since the last backup occurred.
	If it returns any other value, then the return value represents the sum of the return values of the attributes that are set. For example, a return value of 6 represents a hidden system file, where 6 is 2 + 4.

Related Topics

- [Get File Mode Method](#)
- [Set File Attributes Method](#)

Get File Date Method

The Get File Date method returns the last modification date and time of a file.

Format

FileDateTime (*pathname*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string or string expression that evaluates to the name of the file to query. <i>It</i> can contain path and disk information. It cannot include the following wildcards: • * (asterisk) • ? (question mark)

Get File Length Method

The Get File Length method returns the length in bytes of an open file. The file must already be open to use this method.

Format

Lof (*filenumber*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file.

Example

The following example opens a file and prints the contents of this file 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
```

Related Topics

[Erase Array Method](#)

Get File Length 2 Method

The Get File Length 2 method returns the length of a file. If the file is open, then it returns the length of the file before it was opened. The file can be closed or open to use this method.

Format

FileLen(*pathname*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string or string expression that evaluates to the name of the file to query. It cannot include the following wildcards: • * (asterisk) • ? (question mark)

Example

The following 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 file name:"
  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
```

Get File Mode Method

The Get File Mode method returns the file mode or the operating system handle for an open file. The following table describes this return value.

Value of Returntype Argument	Description of the Return Value
1	The file mode of the open file: 1. Input mode. 2. Output mode. 8. Append mode.
2	The operating system handle of the open file.

Format

FileAttr(*filenumber*, *returntype*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file.
<i>returntype</i>	An integer that identifies the type of information to return.

Example

The following example does one of the following:

- If the open file is in input mode or output mode, then it closes this open file.
- If the open file is in append mode, then it writes a range of numbers to this file.

The CreateFile subroutine 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
```

Get File Names Method

The Get File Names method is a standard Visual Basic method that returns the first file name it finds that matches the value in the *pathname* argument and that possesses the attributes that you specify in the attributes argument. If it does not find a file, then it returns a null string ("").

Format

```
Dir[$] [(pathname[, attributes])]
```

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string or string expression that evaluates to a path or file name.

Argument	Description
<i>attributes</i>	An integer expression that specifies the file attributes to choose.

Usage for the Attributes Argument

You can use the integer values for the *attributes* argument described in the following table to return a specific type of file.

Value in Attributes Argument	File Type
0 (default)	Normal files with no attributes set.
2	Normal and hidden files.
4	Normal and system files.
8	Volume label only.
16	Normal files and directories.

You can add values to choose multiple attributes. For example, to return normal files, hidden files, and system files, you set the *attributes* argument to 6, where 6 equals 0 plus 2 plus 4.

If you set the *attributes* argument to 8, then this method returns one of the following values:

- The volume label of the drive that you specify in the *pathname* argument.
- The current drive if you do not specify a drive in the *pathname* argument.

Usage for the Pathname Argument

The *pathname* argument can include a drive specification and the following wildcard characters:

- ? (question mark)
- * (asterisk)

Siebel VB interprets a null string ("") in the *pathname* argument as the current directory. This value is equivalent to a period (.). You can use the Get File Names method again to get more matching file names, but this time do not include the *pathname* argument or the *attributes* argument.

Example

The following example lists all the files that reside on drive A:

```
Sub Button_Click
  Dim msgReturn
  Dim directory, count
  Dim x, msgtext
  Dim A()
```

```
count = 1

ReDim A(100)
directory = Dir ("A:\*.*")
Do While directory <> ""
A(count) = directory
Count = count + 1
directory = 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
```

Get Free File Number Method

The Get Free File Number method returns the lowest unused file number. It does not include arguments. You can use it if you must supply a file number and must make sure that this file number is not already in use. You can use the return value in a subsequent Open statement.

Format

FreeFile

Example

The following example opens a file and assigns to it the next file number that is 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
```

Related Topics

[Open File Method](#)

Lock File Method

The Lock File method controls access to an open file. It does not return a value.

Format

Lock [#]*filenumber*[, [*start*] [*To end*]]

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file.
<i>start</i>	A long integer that identifies the number of the first record or byte offset to lock or unlock.
<i>end</i>	A long integer that identifies the number of the last record or byte offset to lock or unlock.

Specifying the Start Argument and End Argument

The Lock File method locks the file according to the following modes:

- Binary mode. *The start argument and the end argument identify byte offsets.*
- Random mode. *The start argument and the end argument identify record numbers:*
- If you include the *start but not the end*, then it locks only the record or byte that *the start argument identifies*.
- If you *include the end but not the start*, then it locks records or bytes from the record number or offset 1 to the *end*.
- Input mode, output mode, or append mode. *It locks the entire file. It ignores the start argument and the end argument.*

Removing Locks

You must use the Lock File method and the Unlock File method in pairs to remove a lock, and the arguments that you use with these methods must be identical. For more information, see [Unlock File Method](#).

You must configure Siebel VB to remove a lock on an open file before it closes the file. If you do not do this, then unpredictable results might occur.

Example

In the following example, if the file is already in use, then this code locks the file that other users on a network share. The CreateFile subroutine creates the file that the main subroutine uses:

```
(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
btnggrp = 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
```

Open File Method

The Open File method opens a file for input or output. It does not return a value. Siebel CRM opens the file in the default character encoding that the local operating system uses. This method does not support Unicode.

Format

```
Open filename
[For
mode] [Access access] [lock] As [#]filenumber [Len =
reclen]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	A string or string expression that identifies the name of the file to open. If <i>this file</i> does not exist, then this method creates it if opened in one of the following modes: • Append • Binary • Output • Random Modes This file must be open before Siebel VB can perform any input or output operation on it.
<i>mode</i>	A keyword that identifies the purpose for which this method opens the file. If <i>you do not include the mode argument</i>, then <i>this method defaults the mode to random</i>.
<i>access</i>	A keyword that identifies the method to access the file. If <i>you do not include the access argument</i> for random or binary modes, then <i>the Open File method attempts to access the file</i> in the following order: 1. Read Write

Argument	Description
	2. Write 3. Read
<i>lock</i>	A keyword that designates the access method that other processes can use to open this file. If you do not include the <i>lock</i> argument, <i>then other processes can open the file that the filename argument identifies</i> . Other processes cannot perform any file operation on the file while the original process still has the file open.
<i>filenumber</i>	An integer that identifies the file while it is open. You can use the Get Free File Number method to find the next available value that you can use in the <i>filenumber</i> argument. For more information, see Get Free File Number Method .
<i>reclen</i>	In a random or binary file, the length of the records. This method ignores the <i>reclen</i> argument for the following modes: • Input • Output • Append

Usage

The following table describes the keywords that you can use with this method.

Type of Keyword	Keyword	Description
<i>Mode</i>	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.
<i>Access</i>	Read	Reads data only from the file.
	Write	Writes data only to the file.
	Read Write	Reads or writes data to the file.
<i>Lock</i>	Shared	Read or write is available on the file.
	Lock Read	Only read is available.

Type of Keyword	Keyword	Description
	Lock Write	Only write is available.
	Lock Read Write	No read or write is available.

Example

The following example opens a file for random access, gets the contents of the file, and then closes the file. The CreateFile subroutine creates the **c:\temp001** file that the main subroutine uses:

```
(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
```

Rename File Method

The Rename File method renames a file or copies a file from one directory to another directory. It does not return a value. The file that this method renames must be closed. Siebel VB creates an error message in the following situations:

- The file that the *oldfilename argument identifies* is open.
- The file *that the newfilename argument identifies* already exists.

If you use the Rename File method with a Siebel application, and if you do not include the *path2 argument*, then it places a copy of the original file in the following directory under the new name:

c:\siebel\bin

Format

Name [*path1*]*oldfilename* **As** [*path2*]*newfilename*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>path1</i>	A string expression that contains the path to the current file location. If the file is not in the current directory of the current drive, then you must include this argument.
<i>oldfilename</i>	A string expression that contains the name of the file that this method renames.
<i>path2</i>	A string expression that contains the path to the location where the renamed file must reside. If you do not include this argument, then this method saves the file in the current directory of the current drive.
<i>newfilename</i>	A string expression that contains the new name for the file.

Example

The following example creates the **c:\temp001** file, renames this file to **c:\temp002**, and then deletes these files. It calls the **CreateFile** subroutine 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
```

Set File Attributes Method

The Set File Attributes method sets the file attributes for a specified file. It does not return a value.

You cannot use a wildcard in the *pathname* argument. If the file is open, you can modify the file attributes, but only if it is opened for Read access.

Format

```
SetAttr  
    pathname, attributes
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string or string expression that evaluates to the name of the file.
<i>attributes</i>	An integer expression that contains the new attributes of the file.

Usage

The following table describes the attributes you can modify.

Value	Description
0	Normal file.
1	Read-only file.
2	Hidden file.
4	System file.
32	Archive. File has changed since the last backup.

Example

For an example, see [Select Case Statement](#).

Unlock File Method

The Unlock File method controls access to an open file. It does not return a value.

You must use the Lock File method and the Unlock File method in pairs to unlock a file, and the arguments that you use with these methods must be identical. For more information, see [Lock File Method](#).

You must remove a lock before you close the file. For more information, see [Lock File Method](#).

Format

```
Unlock [#]filename[, {record | [start] To end} ]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file. For more information, see Open File Method .
<i>record</i>	An integer that identifies the first record to unlock.
<i>start</i>	A long integer that identifies the first record or byte offset to unlock.
<i>end</i>	A long integer that identifies the last record or byte offset to unlock.

Example

For an example of the Unlock statement, see [Lock File Method](#).

File Input and Output Methods

This topic describes methods that you can use to manipulate file data. It includes the following topics:

- [End of File Method](#)
- [Get Characters From File Method](#)
- [Force Explicit Declaration Statement](#)
- [Get File Offset Method](#)
- [Get File Position Method](#)
- [Get Line From File Method](#)
- [Parse File Contents Method](#)
- [Print Spaces Method](#)
- [Print Data to File Method](#)
- [Set File Position Method](#)
- [Set File Width Method](#)
- [Set Print Position Method](#)
- [Write Data to File Method](#)
- [Write Variable to File Method](#)

End of File Method

The End of File method determines if the end of an open file has been reached. It returns one of the following values:

- -1. The end of the file has been reached.
- 0. The end of the file has not been reached.

Format

Eof(*filenumber*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number you use in the Open statement to open the file. For information about assigning a number to a file when it is opened, see Open File Method .

Example

The following example uses the End of File method to read records from a random file. It keeps the Get statement from attempting to read beyond the end of the file. The CreateFile subroutine creates the **C:\TEMP001** file that the main subroutine uses:

```
(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
```

For another example, see [Get Free File Number Method](#):

Get Characters From File Method

The Get Characters From File method returns a string that contains the characters that it reads from a file. It advances the file pointer according to the number of characters it reads. Unlike the Parse File Contents method, the Get Characters From File method returns every character it reads, including carriage returns, line feeds, and leading spaces.

The input buffer can hold a maximum of 32K characters. If the Get Characters From File method must get an amount of data that exceeds this maximum, then you must call it multiple times, processing 32K characters each time you call it.

Format

`Input[$] (number, [#] filename)`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	An integer that identifies the number of bytes to read from the file.
<i>filename</i>	A number that identifies the open file to use.

Get File Contents Method

The Get File Contents method reads the content of a file opened in random or binary mode, and then places this content in a variable. It does not return a value.

Format

`Get [#] filename, [recnumber], varName`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file. For information about how Siebel VB numbers a file when it opens a file, see Open File Method .
<i>recnumber</i>	An expression of type long. It contains a value that depends on one of the following modes: - Random mode. The record number at which to start reading. - Binary mode. The byte offset at which to start reading. The <i>recnumber</i> argument is in the range of 1 through 2,147,483,647. If you do not include this argument, then this method uses the next record or byte.

Argument	Description
	You must include the commas before and after the <i>recnumber</i> argument even if you do not include the <i>recnumber</i> argument.
<i>varName</i>	The name of a variable. This method reads file data into this variable. It can be any variable type except for the following: • Object. • Array. You can use a single array element.

Usage with Random Mode

For Random mode, the Get File Contents method reads content from the file in chunks whose size is equal to the size specified in the Len clause of the Open statement. It does one of the following, depending on the size of the variable that the *varName* argument identifies:

- The variable is smaller than the record length. It discards the additional content.
- The variable is larger than the record length. It creates an error.

The Get File Contents method handles content differently depending on the following type of variable:

- Variable length string variable. It reads two bytes of content that identifies the length of the string, and then copies the contents into the variable that the *varName* argument identifies.
- Variant variable. It reads two bytes of content that identifies the type of the variant, and then it copies the body of the variant into *the varName* argument. A variant that includes a string includes the following information:

Two bytes of data type information.

Two bytes of length data.

The body of the string.

- Custom variable. It reads the content as if each member were read separately. No padding occurs between elements.

Usage with Binary Mode

Usage with a file opened in Binary mode is the same as usage with a file opened in Random mode except for the following differences:

- The Get File Contents method reads variables from the disk without record padding.
- For a variable length string that is not part of a custom type, the Get File Contents method does not precede this variable with a two-byte string length. Instead, it reads the number of bytes as equal to the length of *the variable that the varName identifies*.

Example

The following example opens a file for random access, gets the contents of this file, and then closes the file. The createfile subroutine creates the **c:\temp001** file that the main subroutine uses:

```
(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
```

Related Topics

[Create Function Method](#)

Get File Offset Method

The Get File Offset method determines the current offset of a file. It returns a value depending on the following mode that it uses to open the file:

- Random file. It returns the number of the last record read or written.
- File opened in append, input, or output mode. It returns the current byte offset divided by 128.
- File opened in binary mode. It returns the offset of the last byte read or written.

The offset starts at 0 in a random file or binary file. A position starts at 1. For more information, see [Get File Position Method](#).

Format

Loc(*filenumber*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file.

Example

The following example creates a file of account numbers that the user enters. When the user finishes, it displays the offset of the file of the last entry that the user 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
```

Get File Position Method

The Get File Position method returns the current position of an open file depending on the following mode that it uses to open the file:

- Random mode. It returns the number of the next record to be read or written.
- Other modes It returns the file offset of 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 number.

Format

Seek(*filenumber*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file. For more information, see Open File Method .

Example

The following example reads the contents of a sequential file line by line to a carriage return, and then displays the results. The CreateFile subroutine creates the **c:\temp001** file that the main subroutine uses:

```
(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
```

Get Line From File Method

The Get Line From File method reads a line from a sequential file, and then saves it in a string variable. It does not return a value. You can use it to read lines of text from a text file where a carriage return separates each data element. You can use a read method to read data from a file that includes values and commas that separate each of these values.

Format A

```
Line Input [#]
    filenumber, varName
```

Format B

```
Line Input [prompt,] varName
```

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	This method does the following depending on if you include the <i>filename</i> argument: • Include <i>filename</i> argument. It uses the file number that the <i>Open</i> statement uses to open the file. • Do not include <i>filename</i> argument. It reads the line from the keyboard.
<i>varName</i>	A string variable. This method saves a line of data or user input into this variable.
<i>prompt</i>	A string literal that prompts the user for keyboard input. If you do not include the <i>prompt argument</i> , then this method displays a question mark (?) as the prompt.

Example

The following example reads the contents of a sequential file line by line up to a carriage return, and then displays the result. The *CreateFile* subroutine creates the **C:\temp001** file that the main subroutine uses:

```
(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
```

Parse File Contents Method

The Parse File Contents method reads data from a sequential file, and then saves this data to different variables. It does not return a value.

Format

Input [#]*filename*, *variable*[, *variable*]...

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file.
<i>variable</i>	One or more variables to contain the values that this method reads from the file. A comma separates each <i>variable</i> .

Example

The following example does the following work:

1. Prompts a user for an account number.
2. Opens a file.
3. Searches the file for the account number.
4. Displays the matching letter for that number.

This example uses the Input statement to increase the value of x and at the same time to get the letter associated with each value. The **CreateFile** subroutine creates the **c:\temp001** file that the main subroutine uses:

```
(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
```

Print Spaces Method

The Print Spaces method prints a number of spaces. It returns a string of spaces in the target of a Print statement. You can use it only in a Print statement. To determine the number of spaces to print, it uses different rules according to the following values:

- The value in the number argument is less than the total line width. It prints the number spaces according to the value of the *number* argument.
- The value in the number argument is greater than the total line width. It prints the number of spaces according to the modulus of the following calculation:

Value in the argument that the Print Spaces method receives divided by the length of the string to print.

The modulus in this context is the remainder of a calculation rounded to an integer.

- X is less than the value of the *number argument* or *number Mod width*, where x is the difference between the current print position and the output line width. It skips to the next line and prints *the value of the number argument* minus x spaces.

You can use the Set File Width statement to set the width of a print line. For more information, see [Set File Width Method](#).

Format

Spc (number)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	An integer or integer expression that specifies the number of spaces to print.

Example

The following example prints five spaces and the following string to a file:

ABCD

This example divides 15 by 10 with a remainder of 5 to determine the five spaces:

```
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
```

Print Data to File Method

The Print Data to File method prints data to an open file. It does not return a value. You can use the following methods in a Print statement:

- Print Spaces method. Inserts a given number of spaces. For more information, see [Print Spaces Method](#).
- Set Print Position method. Moves the print position to a desired column. For more information, see [Set Print Position Method](#).

The Print statement supports only elementary Visual Basic data types. For more information on parsing this statement, see [Get Characters From File Method](#).

Format

```
Print [#][filenumber,] expressionList [{;|, }]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file. The Print Data to File method prints data to this file. For more information, see Open File Method .
<i>expressionList</i>	A list of values that this method prints. It prints these values in the form of literals or expressions. It determines where output for the next Print statement to the same output file must begin. If you do not include the <i>expressionList</i> argument, then this method writes a blank line to the file.

Usage for the Expression List

You can use one of the following characters to separate each value in *the expressionList argument*:

- Semicolon (;). The next value must occur immediately after the preceding value without intervening white space.
- Comma (,). The next value must occur at the next print zone. A new print zone begins every 14 spaces.

If you do not specify a semicolon or a comma, then the Print Data to File method creates a CR-LF (carriage return - line feed) pair and the next Print statement prints on the next line.

Set File Position Method

The Set File Position method sets the position of the next read or write operation in an open file. It does not return a value. If you write to a file after reading beyond the end of the file, then this method increases the file length. If a read operation attempts to specify a negative or zero position, then Visual Basic returns an error message.

Format

`Seek [#]filename, position`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file. For more information, see Open File Method .
<i>position</i>	An expression of type long that identifies the position depending on the following mode that this method uses to open the file: • Random mode. The position of a record number. • Other modes. The position of the byte offset. The <i>position</i> is in the range of 1 through 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

The following example reads the contents of a sequential file line by line to a carriage return, and then displays the results. The CreateFile subroutine creates the **c:\temp001** file that the main subroutine uses:

```
(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

```

Set File Width Method

The Set File Width method sets the output line width for an open file. It does not return a value.

Format

Width [#]*filenumber*, *width*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filenumber</i>	The file number that the Open statement uses to open the file. For more information, see Open File Method .
<i>width</i>	An integer expression that specifies the width of the output line. A value of zero (0) specifies that no line length limit exists. The default value is zero (0).

Example

For an example, see [Print Spaces Method](#)

Set Print Position Method

The Set Print Position method sets the current print position. It does not return a value. You can use it only in a Print statement. Position number 1 is the left most print position. This method sets the new print position according to one of the following values of the position:

- The *value in the position argument* is less than the total line width. It sets the new print position to the value in the *position argument*.
- The *value in the position argument* is greater than the total line width. It sets the new print position according to the following calculation:

The remainder of the input position argument divided by the length of the string

- The current print position is greater than *the position* or the *position Mod width*. It skips to the next line and sets the print position to the value in the *position argument* or to the value in the *position Mod width*.

You can use the Set File Width statement to set the width of a print line. For more information, see [Set File Width Method](#).

Format

Tab (*position*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>position</i>	The position at which Siebel VB begins to print.

Example

The following example prints the octal values for the numbers from 1 through 25. It uses the Set Print Position method to insert 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
```

Write Data to File Method

The Write Data to File method writes data to an open sequential file. It does not return a value. You must open the file in output mode or in append mode. If you do not include the *expressionList* argument, then it writes a blank line to the file. For more information, see [Lock File Method](#).

The Write statement places quotes around the string that it writes to the file.

Format

`Write [#]filename[, expressionList]`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file. For more information, see Open File Method .
<i>expressionList</i>	One or more values to write to the file.

Example

The following example writes a variable to a file according to a comparison of the last saved time of the file 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
```

Related Topics

[Set Print Position Method](#)

Write Variable to File Method

The Write Variable to File method writes a variable to a file opened in random mode or binary mode. It does not return a value. Usage for this method with random mode and binary mode is the same usage for these modes with the Get File Contents method. For more information, see [Get File Contents Method](#) and [Get File Contents Method](#).

The Put statement uses the default character encoding of the local operating system. It does not write to the file in Unicode format.

Format

```
Put [#]filename, [recnumber], varName
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>filename</i>	The file number that the Open statement uses to open the file. For more information, see Open File Method .
<i>recnumber</i>	An expression of type long. It contains a value that depends on one of the following modes: • Random mode. The record number at which to start reading. • Binary mode. The byte offset at which to start reading. The <i>recnumber</i> argument is in the range of 1 through 2,147,483,647. If you do not include this argument, then this method uses the next record or byte. You must include the commas before and after the <i>recnumber</i> argument even if you do not include the <i>recnumber</i> argument.
<i>varName</i>	The name of the variable that contains the data to write. It can be any variable type except for the following types: • Object. • Application data. • Array. You can use a single array element.

Example

The following example opens a file for random access, puts the values 1 through 10 in this file, prints the contents of the file, and then closes it:

```
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
```

Code Setup and Control Methods

This topic describes items that you can use to setup and control Siebel VB code. It includes the following topics:

- [Call Application Method](#)
- [Call Subroutine Method](#)
- [Create Subroutine Method](#)
- [Create Function Method](#)
- [Declare Custom Data Type Method](#)
- [Declare Procedure Method](#)
- [Declare Symbolic Constant Method](#)
- [Get Environment Setting Method](#)
- [Remove Object Method](#)
- [Send Keystrokes Method](#)
- [Use Clipboard Methods](#)

Call Application Method

The Call Application method starts a Microsoft Windows application. It returns the task ID of this application, which is a unique number that identifies the running code.

The pathname argument can contain the name of any valid BAT, COM, EXE, or PIF file. You can include arguments and command line switches. If *the pathname argument does not contain* a valid executable file name, or if the Shell statement cannot start the code, then this method creates an error message.

Format

```
Shell(pathname, [windowStyle])
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	A string or string expression that evaluates to the name of the code to run.

Argument	Description
<i>windowStyle</i>	One of the following integers that specifies how to display the window: • 1. Normal window with focus. • 2. Minimized window with focus. • 3. Maximized window with focus. • 4. Normal window without focus. • 7. Minimized window without focus. If you do not include the <i>windowStyle</i> argument, then Siebel VB uses the default value of 1.

Example

The following example opens Microsoft Excel when the user clicks a button:

```
Sub Button1_Click
  Dim i as long
  i = Shell("C:\Program Files\Microsoft
  Office\Office\EXCEL.EXE",1)
End Sub
```

For other examples, see [Get Right String Method](#) and [Send Keystrokes Method](#).

For more information, see [Send Keystrokes Method](#).

Call Subroutine Method

The Call Subroutine method is a control structure that directs flow to a subroutine or function. It returns one of the following values:

- If it calls a function, then it returns the output of the function.
- if it calls a subroutine, then it returns nothing.

You can use this method to call a subroutine or function that is written in Visual Basic or to call C code in a DLL. A Declare Procedure method must describe this C code and it must be implicit in the application. You must make sure the DLL is present on every Siebel Server. For more information, see [Declare Procedure Method](#).

If you use the Call Subroutine method, then it is recommended that you use the following guidelines:

- [Pass Values Through Reference](#)
- [Give Each Argument a Name](#)

Format A

```
Call subroutine_name [(argument_list)]
```

Format B

```
subroutine_name argument_list
```

where:

- *subroutine_name* is the name of the subroutine or function. Siebel VB passes control to this subroutine or function.

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>argument_list</i>	The arguments that Siebel VB passes to the subroutine or function.

Example

The following example does the following:

1. Calls a subroutine named **CreateFile** to open a file.
2. Writes the numbers 1 through 10 in this file.
3. The calling code examines the file mode, and then closes the file if the mode is 1 or 2:

```
(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
```

Related Topics

[Declare Procedure Method](#)

Create Subroutine Method

The Create Subroutine method is a control structure that defines a subroutine. It does not return a value. It returns flow to the caller when Siebel VB encounters the End Sub statement or an Exit Sub statement.

Format

```
[Static] [Private] Sub name [(Optional) argument [As  
    type], ...)]  
End Sub
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>name</i>	The name of the subroutine.
<i>argument</i>	A list of argument names that includes commas that separate each name. For more information, see the following.
<i>type</i>	The data type of the <i>argument</i> .

Usage

Note the following:

- For important caution information, see [Create Function Method](#).
- A call to a subroutine stands alone as a separate statement. For more information, see [Call Subroutine Method](#).
- Siebel VB supports recursion.
- A Visual Basic procedure passes values through reference. This means that if a procedure assigns a value to an argument, then it modifies the variable that the caller passes. You must use this feature with caution. For more information, see [Pass Values Through Reference](#).
- You must use the Create Function method rather than the Create Subroutine method to define a procedure that includes a return value. For more information, see [Create Function Method](#).

Usage for the Static Keyword and Private Keyword

The Static keyword specifies that any variable you declare in this method retains a value as long as the code runs. This situation applies regardless of how you declare variables in this method.

The Private keyword specifies that other functions and subroutines in other modules cannot access the subroutine that you define with this method. Only procedures defined in the same module can access a private function.

Specifying Arguments

Note the following:

- To specify multiple arguments, you can use a list of variable names where a comma separates each name.
- To specify the data type of an argument, you can use a type character or the As clause.
- To declare a record argument, you use the As clause and a value in the *type argument*. You must have already used the Type statement to define this value in the type argument.
- To declare an array argument, you can use empty parentheses after the *argument*. You do not specify array dimensions in the Create Subroutine method. You must use a consistent number of dimensions for every reference to an array argument that occurs in the body of the Create Subroutine code.

Declaring Optional Arguments

If you declare an optional *argument*, then you can omit the value for this argument when Siebel VB calls the method that contains this argument. Note the following:

- You can only declare a variant argument as optional.
- Any optional arguments must occur after the required arguments in the Create Subroutine method.
- You can use the IsMissing method to determine if an optional argument is omitted. For more information, see [Is Optional Argument Missing Method](#). For more information on using named arguments, see [Comments](#) and [Call Subroutine Method](#).

Example

The following example is a subroutine that uses the Create Subroutine method:

```
Sub Button1_Click
End Sub
```

Related Topics

- [Declare Variable Statement](#)
- [Create Function Method](#)
- [Declare Global Variable Statement](#)
- [Force Explicit Declaration Statement](#)
- [Declare Global Variable Statement](#)

Create Function Method

The Create Function method creates a function. It returns to the caller when it encounters an End Function statement or an Exit Function statement. It returns the value that *the expression argument calculates*.

Format

```
[Static] [Private] Function name([[Optional ]argument  
[As type]][, ... ]) [As funcType]  
    name = expression  
End Function
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>name</i>	The name of the function.
<i>argument</i>	The argument to pass to the function when Siebel VB calls it. For more information, see Create Subroutine Method .
<i>type</i>	The data type of the argument.

Argument	Description
<i>funcType</i>	The data type of the value that the function returns.

Usage

For information about the static and private keywords, see [Create Subroutine Method](#).

A Visual Basic procedure passes values through reference. This means that if a procedure assigns a value to an argument, then it modifies the variable that the caller passes. You must use this feature with caution. For more information, see [Pass Values Through Reference](#).

For information about declaring optional arguments, see [Create Subroutine Method](#).

Specifying the Type

Specifying the type character when you use the Create Function method is optional. A function can create and return a single value of a type that you specify. The data type of the value in the *name argument* determines the type of the return value. You can do one of the following to specify the data type:

- Use a type character as part of the *name*.
- Use the *As funcType* clause.

If you do not use one of these formats, then Siebel VB uses the default data type, which is variant.

Specifying the Return Value

You use the following format to specify the return value for the function name:

```
name = expression
```

where:

- *name* is the name of the function
- *expression* evaluates to a return value

If you do not include this code, then Siebel VB returns one of the following values:

- 0 for a numeric function
- null string ("") for a string function
- An Empty variable type for a return type of variant. For more information, see [Variants](#).

You can use the Sub statement to define a procedure that does not include a return value.

Caution About Writing a Custom Function or Subroutine

If you create more than one function or subroutine in the general declarations section, then you must make sure that any other custom function or subroutine that you create that calls this function or subroutine occurs before the procedure that calls it. Otherwise, you cannot compile your procedures.

You cannot create a custom function or custom subroutine in a method or event that displays in Siebel Tools. You can create a custom function or custom subroutine in the general declarations section in the script of a method. If your function or subroutine must be available throughout the code, then you can use a `PreInvokeMethod` method or an external DLL file to store them in a central location. For more information, see doc ID 476501.1 on My Oracle Support.

Example

The following example declares a function that the subroutine calls. The function performs a calculation on the value sent to it. This calculation modifies the value of the variable:

```
(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
```

For other examples, see [Declare Procedure Method](#), and [Go To Statement](#).

Related Topics

- [Call Subroutine Method](#)
- [Create Subroutine Method](#)
- [Declare Variable Statement](#)
- [Declare Global Variable Statement](#)
- [Force Explicit Declaration Statement](#)
- [Is Optional Argument Missing Method](#)

Declare Custom Data Type Method

The `Declare Custom Data Type` method declares a custom data type. It does not return a value. For more information, see [About Data Types](#).

Format

```
Type
    userType
    field1 As type1
    field2 As type2
    ...
End Type
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>userType</i>	The name of the custom data type.
<i>field1, field2</i>	Each argument specifies the name of a field in the custom type.
<i>type1, type2</i>	Each argument specifies the data type of the field.

Usage

You can use the following format to access the fields of a record:

recordName.fieldName

You can use the following format to access the fields of an array of records:

arrayName(index).fieldName

You cannot specify an array in the field argument. Arrays of records are allowed.

You cannot use the Type statement in a procedure definition. You must use it in the general declarations section. For example, see [Set Array Lower Boundary Method](#).

Siebel VB cannot pass a custom type to a COM function or subroutine.

To declare a record variable, you can use the Declare Custom Data Type method in a Declare Variable method. Siebel VB does not allocate memory when you define a custom type. It only allocates memory if you use a Declare Variable method to declare a custom type. If you declare a variable of a custom type, then you are *instantiating* the type. For more information, see [Declare Variable Statement](#).

Example

The following example includes a Type statement and a Dim statement. You must define a record type before you can declare a record variable. The subroutine references a field in 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.custncustomame = "Chris Smith"
  If myrecord.custname = "" then
    Exit Sub
  End If
End Sub
```

Related Topics

- [Set Variable Data Type Statement](#)
- [Declare Variable Statement](#)

Declare Procedure Method

The Declare Procedure method declares a procedure in a module or in a dynamic link library (DLL). The value it returns depends on one of the following formats that you use:

- Format A. It does not return a value.
- Format B. A value of the funcType type. You can use this value in an expression.

For more information about the format and arguments that you can use with this method, see [Declaring a Procedure](#).

Format A

```
Declare Sub name [(argument [As type])]
```

Format B

```
Declare Function name [(argument [As type])] [As funcType]
```

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>name</i>	The name of the subroutine or function that this method declares.
<i>argument</i>	The argument that this method passes. A comma separates each argument.
<i>type</i>	The data type of the arguments.
<i>funcType</i>	The data type of the return value.

Example

The following example declares a function that the main subroutine subsequently calls. This function only sets the return value to 1:

```
(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
```

For other examples of functions, see [Create Function Method](#) and [Go To Statement](#).

Related Topics

- [Create Function Method](#)
- [Declare Symbolic Constant Method](#)
- [Declare Variable Statement](#)
- [Declare Global Variable Statement](#)
- [Set Variable Data Type Statement](#)

Declare Symbolic Constant Method

The Declare Symbolic Constant method declares a symbolic constant. It does not return a value.

Format

```
[Global] Const constantName [As type] = expression [, constantName [As type] =  
expression] ...
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>constantName</i>	The variable name to contain a constant value.
<i>type</i>	The data type of the constant. This type is Number or String.
<i>expression</i>	Any expression that evaluates to a constant number.

Usage

To specify the type of the constant, you can use one of the following characters as a suffix of the *constantName* argument:

- # (pound sign). Specifies a number.
- \$ (*dollar sign*). Specifies a string.

You can use this technique instead of using the As clause.

If you do not specify a type character, then the Declare Symbolic Constant method derives the type of the value that you specify in the *constantName* argument from the type of the expression.

To specify a global constant, you must declare it in the general declarations section of the module where you must access this global variable.

Example

For an example, see [Convert Expression to Long Method](#).

Get Environment Setting Method

The Get Environment Setting method returns the string setting that is assigned to an environment variable for a keyword in the environment table of the operating system. If it cannot find the value that you specify, then it returns a null string.

Format A

Environ[\$] (*environment-string*)

The return value for format A is the string that is associated with the keyword.

For information about the dollar sign, see [Usage of the Dollar Sign](#).

Format B

Environ[\$] (*numeric_expression*)

The return value for format B is a string that uses the following format:

KEYWORD=value

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>environment-string</i>	The name of a keyword in the operating system. You must enter the value for this argument in uppercase. If you do not, then this method returns a null string ("").
<i>numeric_expression</i>	An integer for the position of the string in the environment table. For example, 1st, 2nd, 3rd, and so on. This method rounds the numeric expression to a whole number, if necessary.

Example

The following 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
```

Remove Object Method

You can use the Remove Object method to remove from memory an object that Siebel VB instantiates in memory. If an object variable does not reference an object, then it contains a value of Nothing. The Remove Object method does not return a value.

Format

```
Set
  objectName = Nothing
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>objectName</i>	The name of the object variable to set to Nothing.

For example:

```
If Not
  objectVar
Is Nothing then

  objectVar
  .Close
Set

  objectVar
  = Nothing
End If
```

Example

The following example adds an activity record when the user adds a contact record in a Siebel application. It presumes that the Contact business component is the parent business component. It instantiates the Action business component, and then uses the Remove Object method to remove this instance from memory after it finishes processing the business component data:

```
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
```

For other examples that use the Remove Object method, see [Date and Time Methods](#) and [Get COM Object Method](#).

Send Keystrokes Method

The Send Keystrokes method sends keystrokes to an active Microsoft Windows application. It does not return a value. It can send a keystroke only to the currently active application. You can use the AppActivate statement to activate an application. You cannot use this method to send keys to an application that does not run in Microsoft Windows.

Format

```
SendKeys  
string[, wait]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression that contains the characters to send. Each character in the <i>string</i> argument represents a keystroke.
<i>wait</i>	Specifies to wait until Siebel VB processes every key before it continues to the next code line. It can include one of the following values: -1. Wait. 0. Do not wait. This is the default value.

Specifying Alphanumeric Characters

To specify an alphanumeric character, you enter it in the *string* argument. For example, to send the character *a*, you use **a** as the value in the *string* argument. You can combine multiple characters in one string. For example, if the *string* argument contains **abc**, then Siebel VB sends the following values to the application:

a, *b*, and *c*

Specifying Control Keys

To specify that the user must press the SHIFT, ALT, or CTRL key simultaneously with a character, you prefix the character with one of the following values:

- +. Specifies SHIFT.
- %. Specifies ALT.
- ^. Specifies CTRL.

You can use parentheses to specify that the user must press the SHIFT, ALT, or CTRL key with a group of characters. For example, **%(abc)** is equivalent to **%a%b%c**.

Control Keys That the Send Keystrokes Method Interprets

The following table describes some keys that the Send Keystrokes method interprets. To specify one of these characters as a literal value, you must enclose the character in curly brackets ({}).

Key	Description
+	SHIFT key.
%	ALT key.
^	CTRL key.
()	Apply a shift state to the characters that the parentheses enclose.
~	Newline. Note the following: • Tilde (~). Represents the ENTER key on an alphanumeric keypad. • {Enter}. Represents the ENTER key on a numeric keypad.
{ }	Makes the enclosed characters literals.
[]	Square brackets do not possess a special meaning for the Send Keystrokes method, but they might possess a special meaning for other applications.

For example, if the *string argument* contains **{%}**, then the Send Keystrokes method sends the literal value of the percentage symbol (%).

You can use the following format to send a bracket:

- Left bracket. You use **{}**.
- Right bracket. You use **{}.**

Repeating the Same Key

To send the same key multiple times, you can enclose the character in curly brackets ({}), add a space, and then specify the number of keys to send. For example, the following code sends 20 X characters:

```
{X 20}
```

Sending Nonprintable Keys

The following table describes how you can to send a nonprintable key. You can use a special keyword and enclose it in curly brackets ({}).

Nonprintable Key	Format
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}
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}
Function key (F1 through F15)	You enclose the name of the function key in curly brackets ({}). For example, to send F5, you use the following format:

Nonprintable Key	Format
	{F5}

Combining Keywords

You can use the following characters to combine some keywords:

- Plus sign (+)
- Percentage symbol (%)
- Caret (^)

For example, **%{TAB}** is ALT+TAB.

You can send multiple keywords. For example, the following code sends 25 up arrows:

```
{UP 25}
```

Example

The following example starts the Microsoft Windows Phone Dialer application and dials a phone number that the user enters:

```
Sub Button_Click
  Dim phonenumber, msgtext
  Dim x
  phonenumber = 650-555-1212
  x = Shell ("Terminal.exe",-1)
  SendKeys "%N" & phonenumber & "{Enter}", -1
End Sub
```

Related Topics

Call Application Method

Use Clipboard Methods

The Use Clipboard methods are standard Visual Basic methods that allow you to use the Microsoft Windows clipboard as an object. They do not return a value. You can use the Microsoft Windows Clipboard to transfer text to and from other applications that support this clipboard.

Format

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

The following table describes the arguments that you can use with these methods.

Argument	Description
<i>string</i>	A string or string expression that contains the text to send to the clipboard.

The following table describes the statements that you can use the clipboard.

The following table describes the arguments that you can use with these methods.

Statement	Description
Clear	Clears the contents of the clipboard.
GetText	Returns a text string from the clipboard.
SetText	Puts a text string in the clipboard.
GetFormat	Returns one of the following values: • Not zero. The format of the item on the clipboard is text. • Zero (0). The format of the item on the clipboard is not text.

If code or if a cut or copy operation places data that is of the same format as the data that currently resides in the clipboard, then the current data on the clipboard is lost.

Example

The following example places the following text string on the clipboard:

Hello, world:

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

Code Control Statements

This topic describes statements that you can use to control the flow of Siebel VB code. It includes the following topics:

- *Do Loop Statement*
- *Exit Statement*
- *For Next Statement*
- *Go To Statement*
- *If Then Else Statement*
- *Go To Label Statement*

- *Me Statement*
- *Rem Statement*
- *Select Case Statement*
- *Stop Statement*
- *While Wend Statement*

Do Loop Statement

The Do Loop statement is a control structure that repeats a series of code lines as long as an expression is TRUE. It does not return a value.

If an Exit Do statement runs, then control flows to the statement that resides immediately after the Loop statement. If you use an Exit Do statement in a nested loop, then it moves control out of the immediate loop.

Format A

```
Do [{ While|Until } condition]
statement_block
[Exit Do]
statement_block
Loop
```

Format B

```
Do
statement_block
[Exit Do]
statement_block
Loop [{ While|Until } condition]
```

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>condition</i>	Any expression that evaluates to TRUE (not zero) or FALSE (0).
<i>statement_block</i>	Code lines to repeat while or until the value you specify in the <i>condition argument</i> is TRUE (not zero).

Example

For examples, see *Get File Names Method*, *Erase Array Method*, and *Get Error Code Method*.

Exit Statement

The Exit statement is a control structure that stops statements that reside in a loop or transfers control to a calling procedure. It does not return a value.

You can include an Exit Do statement in a Do Loop statement. You can use an Exit For statement in a For Next statement. When the Exit statement runs, control transfers to the statement that occurs after the Loop statement or the Next statement. When used in a nested loop, an Exit statement moves control out of the immediately enclosing loop.

Note the following:

- You can use the Exit statement in the Create Function method. For more information, see [Create Function Method](#).
- You can use the Exit Sub statement in the Create Subroutine method. For more information, see [Create Subroutine Method](#).

Format

```
Exit {Do | For | Function | Sub}
```

Example

The following example uses the On Error statement to handle run-time errors. If an error exists, then the code continues at the Debugger label. This example uses the Exit statement to skip the debugging code when no error exists:

```
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
```

For Next Statement

The For Next statement is a control structure that repeats a series of code lines a fixed number of times. It does not return a value.

Format

```
For counter = start To end [Step increment]
```

```
  statement_block
```

```
[Exit For]
```

```
  statement_block
```

```
Next [counter]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>counter</i>	A numeric variable for the loop counter.
<i>start</i>	The initial value of the counter.
<i>end</i>	The ending value of the counter.
<i>increment</i>	The amount by which Siebel VB modifies the counter each time the loop runs. The default value is 1.
<i>statement_block</i>	The Visual Basic functions, statements, or methods to run.

How Siebel VB Handles a For Next Statement

Siebel VB compares the sign of *start* and the sign of *end* to the sign of *increment*, and then does one of the following:

- If the signs are the same, and if *end* does not equal *start*, then it starts the For Next loop.
- If the signs are not the same, then it skips the loop entirely.

Siebel VB does the following when it runs a For Next loop:

1. Runs the code lines that occur after the For statement until it encounters the Next statement.
2. Adds the Step amount to the counter.
3. Compares the value in the counter argument to the value in the end argument.
4. Does one of the following:
5. If the beginning and ending values are the same. It runs the loop one time, regardless of the Step value.
6. If the beginning and ending values are not same. It uses the Step value to control the loop as follows:

Step Value	Description
Positive	Siebel VB does one of the following, depending on the value of the counter: • The <i>counter</i> is less than or equal to the value in the <i>end argument</i>. It adds the Step value to <i>counter</i>. Control returns to the statement after the For statement and the process repeats. • The <i>counter</i> is greater than the value in the <i>end argument</i>. It exits the loop, and then resumes running code with the statement that occurs immediately after the Next statement.
Negative	Siebel VB repeats the loop until <i>the counter</i> is less than <i>the value in the end argument</i> .
Zero	Siebel VB repeats the loop indefinitely.

Usage

The values in the *start argument* and the *end argument* must be consistent with *the value in the increment argument*:

- If *end* is greater than *start*, *then increment* must be positive.
- If *end* is less than *start*, *then increment* must be negative.

You must not modify the value of the *counter* while the loop runs. Changing this value makes the code more difficult to maintain and debug.

You can use the Exit For statement as an alternative exit from a For Next loop.

Nesting a For Next Loop

You can nest a For Next loop in another For Next loop:

- You must specify a unique variable name for the *counter* in each nested loop.
- You must make sure the Next statement of the inner loop occurs before the Next statement of the outer loop.

To merge Next statements that are multiple and consecutive, you must make sure the innermost counter occurs first and the outermost counter occurs last. For example:

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

If you do not include the variable in a Next statement, then this Next statement matches the most recent For statement. If a Next statement occurs prior to the corresponding For statement for this Next statement, then Siebel VB returns an error message.

Example

For an example, see [Convert Expression to Single-Precision Method](#).

Go To Statement

The Go To statement is a control structure that directs flow to a label. It does not return a value. The value of the label argument uses the same format as any other Visual Basic name. A reserved word is not a valid label. You cannot use a Go To statement to transfer control out of the current function or subroutine. It is recommended that you avoid using a Go To statement. For more information, see the following.

Format

```
GoTo
label
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>label</i>	A name that begins in the first column of a line of code and ends in a colon (:).

Example 1

The following example displays the date for one week from the date that the user enters. If the date is not valid, then the Go To statement directs flow to the beginning of the start statement:

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

Example 2

It is recommended that you avoid using a Go To statement. When possible, you must use another technique. For example, *Go To Statement* could use an If statement that occurs in a separate function that the main code calls. If the test fails, then Siebel VB can call the initial code again. For example:

```
(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 = CDate(str1)
  nextweek = DateValue(str1) + 7
  CheckResponse = "One week from the date entered is " & _
    Format(nextweek, "ddddd")
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
```

If Then Else Statement

The If Then Else statement is a control structure that runs a block of code according to one or more expressions. It does not return a value.

Format A

```
If  
  condition  
  Then  
    then_statement [Else  
    else_statement]
```

Format B

```
If condition Then  
  statement_block  
  [ElseIf  
  expression  
  Then  
    statement_block ]...  
  [Else  
    statement_block ]  
End If
```

If you require multiple statements in the Then clause or in the Else clause, then you must use Format B.

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>condition</i>	Any expression that evaluates to TRUE (not zero) or FALSE (zero).
<i>then_statement</i>	Any valid single expression.
<i>else_statement</i>	Any valid single expression.
<i>expression</i>	Any expression that evaluates to TRUE (not zero) or FALSE (zero).
<i>statement_block</i>	Zero or valid expressions where a colon (:) or a different code line separates each expression.

Example

The following example examines the time and the day of the week and returns a 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
```

Go To Label Statement

The Go To Label statement is a control structure that directs flow to a label in the current procedure according to the value of a numeric expression. It does not return a value.

If the value in the *number argument* evaluates to:

- Zero or to a number that is greater than the number of labels that occur after the Go To Label statement, then Siebel VB runs the code at the next statement.
- Less than 0 or greater than 255, then Siebel VB creates the following error:

```
Illegal function call
```

Format

```
On
number GoTo label1[, label2, ... ]
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	Any numeric expression that evaluates to a positive number.
<i>label1, label2, ...</i>	A label in the current procedure. Siebel VB directs flow to this procedure.

Me Statement

The Me statement is standard Visual Basic shorthand that refers to the currently used object. It does not return a value. A Siebel VB module can be attached to an application object. If this application object encounters a some events, then Siebel VB might call a subroutine. For example, if the user clicks a button, then the Me statement runs Visual Basic code or a statement calls a method on an application object.

A subroutine in this situation can use the Me variable to reference the object that starts the event. For example, the button click. You can use the Me statement the same way that you use any other object variable except that you can use the Assign COM Object statement to set Me.

Format A

```
With Me
    .methodname() statement
End With
```

Format B

```
Me.methodname() statement
```

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>methodname</i>	The name of the method that Siebel VB uses with the object.
<i>statement</i>	The statement argument includes one of the following items: • The code that Siebel VB runs • The arguments to the method

Example

For an example, see [Modify Variable Statement](#).

Related Topics

- [Date and Time Methods](#)
- [Get COM Object Method](#)
- [Initialize COM Object Method](#)
- [Remove Object Method](#)
- [COM Object Class](#)
- [Is Object Of Class Method](#)

Rem Statement

The Rem statement identifies a line of code as a comment in Visual Basic code. It does not return a value. A code line that begins with a single quote (') also identifies a comment.

Format

```
Rem
    comment
```

Example

The following example code 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 forth, 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 successfully
  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
```

Select Case Statement

The Select Case statement is a control structure that runs one or more statements, depending on the value of an expression. It does not return a value.

Format

```
Select Case testexpression
  Case expressionList
    [statement_block]
  [Case expressionList
    [statement_block] ]
  .
  .
  [Case Else
    [statement_block]
  End Select
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>testexpression</i>	Any expression that contains a variable.
<i>expressionList</i>	One or more expressions that contain a possible value for the <i>testexpression</i> argument.
<i>statement_block</i>	One or more lines of code to run if the value in the <i>testexpression</i> argument equals a value in the <i>expressionList</i> argument.

Usage

If the value in the *testexpression* argument is equal to the value in the *expressionList* argument, then Siebel VB runs the *statement_block* that occurs immediately after the Case clause. When Siebel VB encounters the next Case clause, the code flows to the statement that occurs immediately after the End Select statement.

The *expressionList* can include a list of expressions. A comma separates each expression in this list. It can include one of the following forms:

expression

expression To expression

Is *comparison_operator* *expression*

The type of each *expression* must be compatible with the type of the *testexpression*.

Each *statement_block* can contain any number of statements on any number of lines.

Using the Is Keyword

If you use the To keyword to specify a range of values, then the smaller value must occur first. The *comparison_operator* that you use with the Is keyword must contain one of the following:

- < (less than)
- > (greater than)
- = (equal)
- <= (less than or equal)
- >= (greater than or equal)
- <> (not equal)

If the Case is one end of a range, then you must use the Is operator. For example:

Case Is < 100

Stop Statement

The Stop statement is a control structure that stops code from running. It does not return a value. It does not include arguments. You can include a Stop statement anywhere in Siebel VB code. A Stop statement does not close files or clear variables.

Format

Stop

Example

The following example stops code from running when the user clicks a button:

```
Sub Button_Click
  Dim str1
  str1 = Y
  If str1 = "Y" or str1 = "y" then
    Stop
  End If
End Sub
```

While Wend Statement

The While Wend statement is a control structure that controls a repetitive action. It does not return a value. Siebel VB includes the While statement so that it is compatible with older versions of Visual Basic. If possible, it is recommended that you use the Do Loop statement instead. For more information, see [Do Loop Statement](#).

Format

```
While
  condition
  statement_block
Wend
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>condition</i>	A condition where Siebel VB runs the statements in a statement_block.
<i>statement_block</i>	A series of statements that Siebel VB runs while <i>the value in the condition argument</i> is TRUE.

Example

The following example opens a series of files and looks for the following string in each file:

Overdue

This example uses the While Wend statement to loop through the **c:\temp00?** files. The CreateFiles subroutine creates these files:

```
(general) (declarations)
Option Explicit
Declare Sub CreateFiles
Sub CreateFiles
    Dim oduo as String
    Dim ontime as String
    Dim x
    Open "c:\temp001" for OUTPUT as #1
    oduo = "*Overdue*"
    ontime = "*On-Time*"
    For x = 1 to 3
        Write #1, oduo
    Next x
    For x = 4 to 6
        Write #1, ontime
    Next x
    Close #1
    Open "c:\temp002" for Output as #1
    Write #1, oduo
    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$("temp00?")
    pattern = "*" + "Overdue" + "*"
    While custfile <> ""
        Open custfile for input as #1
        On Error goto atEOF
        Do
            Line Input #1, aline
            If aline Like pattern Then
                count = count + 1
            End If
        Loop
    nextfile:
        On Error GoTo 0
        Close #1
        custfile = Dir$
        Wend
        Kill "c:\temp001"
        Kill "c:\temp002"
    Exit Sub
atEOF:
    Resume nextfile
End Sub
```

Variable Manipulation Methods

This topic describes statements and methods that setup and control variables. It includes the following topics:

- *Assign Expression to Variable Statement*

- *Declare Variable Statement*
- *Declare Global Variable Statement*
- *Declare Static Variable Statement*
- *Modify Variable Statement*
- *Force Explicit Declaration Statement*
- *Get Variant Type Method*
- *Set Variable Data Type Statement*
- *Set Variant Variable to Null Method*

Assign Expression to Variable Statement

The Assign Expression to Variable statement is a predefined VB statement that assigns an expression to a Visual Basic variable. It does not return a value. You can use it to assign a value or expression to a variable that is of one of the following data types:

- Numeric
- String
- Variant
- Record

If you use this statement to assign a value to a numeric variable or to a string variable, then Siebel VB applies the standard conversion rules.

You can use this statement to assign a value to a field in a record or to assign a value to an element in an array.

The Let keyword is optional. Let is different from Set because 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.

Format

```
[Let]  
variable = expression
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variable</i>	The variable where Siebel VB assigns a value.
<i>expression</i>	The expression that contains the value that Siebel VB assigns.

Example

The following example uses the Assign Expression to Variable statement for the sum variable. It calculates 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
```

Related Topics

[Declare Symbolic Constant Method](#)

Declare Variable Statement

The Declare Variable statement declares a variable. It does not return a value. For more information about the format and arguments you can use this statement, see [Declaring Variables](#).

Dim is an abbreviation for Declare in Memory. *You must begin the value in the VariableName argument with a letter.* It must contain only letters, numbers, and underscores. You can use square brackets to separate a name. You can use any character between the square brackets except for more square brackets. For example:

```
Dim my_1st_variable As String
Dim [one long and strange! variable name] As String
```

Format

```
Dim [Shared] variableName [As[ New] type] [, variableName [As[ New] type]] ...
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variableName</i>	The name of the variable to declare.
<i>type</i>	The data type of the variable.

Example

The following example includes 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
```

Related Topics

- [Create Function Method](#)
- [Declare Array Method](#)
- [Set Array Lower Boundary Method](#)

Declare Global Variable Statement

The Declare Global Variable statement declares a global variable. It does not return a value. You must declare a global variable in every module from which Siebel VB must access that variable. You declare a global variable in the general declarations section of the module.

Format

```
Global
variableName
[As
type] [,variableName [As type]] ...
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variableName</i>	The variable name.
<i>type</i>	The data type of the variable.

Usage

If you do not include the As clause, then you can add a type character as a suffix to the *variableName* argument. You can simultaneously use the two different type specification methods in a single Global statement, but you cannot use these methods simultaneously on the same variable.

Regardless of how you declare a global variable, you can choose to include or not include the type character when you reference the variable from another section of code. Siebel VB does not consider the type suffix as part of the variable name.

Formats That You Can Use to Specify the Type of a Global Variable

Visual Basic is a strongly typed language. You must assign a data type to a variable or Siebel VB assigns a type of variant.

The following table describes the data types you can use to specify the type of a global variable. Declaring a global variable is the same as declaring a variable, except where noted in the Format column in the table. The Reference column includes a link to the description for declaring a variable.

Type	Format	Reference
Array	You use the following format to declare a global record: <pre> Global variable ([subscriptRange , ...]) [As typeName]</pre> where: <i>subscriptRange</i> uses the following format: <pre> [startSubscript To] endSubscript</pre>	Declaring an Array Variable
Number	Not applicable.	Declaring a Number Variable
Record	You use the following format to declare a global record: <pre> Global variableName As typeName</pre> You cannot use the Declare Global Variable statement to declare a dialog record.	Declaring a Record Variable
String	You use the following format to declare a global string: <pre> Global variableName As String * length</pre> You use one of the following formats to declare a dynamic string:	Declaring a String Variable

Type	Format	Reference
	Global <i>variableName</i> \$ Global <i>variableName</i> As String	
Variant	You use one of the following formats to declare a global variant: Global <i>variableName</i> Global <i>variableName</i> As Variant	<i>Declaring a Variant Variable</i>

Example

The following example includes two subroutines that share the *total* and *acctno* variables, and the *grecord* record:

```
(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
```

Related Topics

- [Create Function Method](#)
- [Declare Symbolic Constant Method](#)
- [Declare Array Method](#)
- [Set Array Lower Boundary Method](#)

Declare Static Variable Statement

The Declare Static Variable statement declares a variable and allocates storage space for this variable. It does not return a value.

A variable that you declare with the Static statement retains a value as long as the code runs. The format you use is exactly the same as the format you use with the Declare Variable statement. For more information, see [Declare Variable Statement](#).

To make a procedure variable static, you can use the Static keyword in the definition of this procedure. For more information, see [Create Function Method](#) and [Create Subroutine Method](#).

Format

Static *variableName* [As *type*] [,*variableName* [As *type*]] ...

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variableName</i>	The name of the variable to declare as static.
<i>type</i>	The data type of the variable. If you do not include the type argument, then Siebel VB uses the variant type.

Related Topics

- [Declare Array Method](#)

- [Set Array Lower Boundary Method](#)

Modify Variable Statement

The Modify Variable statement runs a series of statements on a variable. It does not return a value. *The value you specify in the variable argument* can be an object or a custom type. You can nest With statements.

Format

```
With  
    variable  
    statement_block  
End With
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variable</i>	The variable that Siebel VB modifies in the <i>statement_block</i> .
<i>statement_block</i>	The statements that Siebel VB runs on the variable.

Example

The following example uses a Siebel VB method to modify the values that an object contains if Siebel CRM modifies a field value. The Modify Variable statement references this object:

```
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
```

For another example, see [Remove Object Method](#).

Force Explicit Declaration Statement

The Force Explicit Declaration statement forces you to explicitly declare every variable in a module. It does not return a value.

Format

```
Option Explicit
```

Usage

Visual Basic declares any variables that does not occur in one of the following statements, by default:

- Dim
- Global
- ReDim
- Static

If you include the Force Explicit Declaration statement in your code, then Siebel VB creates the following error every time it encounters a variable that is not declared:

```
Variable Not Declared
```

Using the Force Explicit Declaration statement makes debugging code easier because it forces you to declare each variable before you use it. It is recommended that you declare variables at the beginning of the project, module, or procedure where these variables possess scope. Declaring variables in this way simplifies locating their definitions when reading through code.

You must include the Force Explicit Declaration statement in the general declarations section. For more information, see [Set Array Lower Boundary Method](#).

Example

The following example specifies that variables must be explicitly declared. This technique prevents 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
```

Related Topics

- [Create Subroutine Method](#)
- [Create Function Method](#)
- [Declare Symbolic Constant Method](#)
- [Declare Array Method](#)

Get Variant Type Method

The Get Variant Type method returns a number that represents the type of data stored in a variant variable. For more information, see [Variants](#).

Format

VarType (*varName*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>varName</i>	The name of a variant variable.

Example

The following example returns the variant type of the myarray variant variable:

```
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
```

Related Topics

- [Is Expression a Date Method](#)
- [Is Variable Null Method](#)
- [Is Variable Numeric Method](#)
- [Is Variable Set Method](#)

Set Variable Data Type Statement

The Set Variable Data Type statement sets the default data type for one or more variables.

Format

```
DefCur varTypeLetters
DefInt varTypeLetters
DefLng varTypeLetters
DefSng varTypeLetters
DefDb1 varTypeLetters
DefStr varTypeLetters
DefVar varTypeLetters
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>varTypeLetter</i>	The first letter of a variable name.

Usage

The *VarTypeLetters* argument can be one of the following:

- Single letter
- List of letters that includes a comma to separate each letter
- Range of letters

For example, if you specify a-d, then Siebel VB uses letters a, b, c, and d.

The case of the letters is not important, even in a letter range. Siebel VB considers the a-z letter range as all alpha characters, including international characters.

The *Deftype* statement affects only the code where you specify it. It must precede any variable definition in the code.

To override the `Deftype` statement, you can use the `Global` statement or the `Declare Variable` statement to declare a variable, and then use an `As` clause or a type character.

For more information, see [Variants](#).

Example

The following example determines the average of bowling scores that the user enters. Because the **average** variable begins with A, this example defines it as a single-precision, floating point number. It defines the other variables 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
```

Related Topics

- [Create Function Method](#)
- [Declare Procedure Method](#)

Set Variant Variable to Null Method

The `Set Variant Variable to Null` method sets a variant variable to a value of `Null`. It returns a variant that is set to `NULL`. Note that Visual Basic sets a variant to the empty value, which is different from the `Null` value.

Format

`variableName = Null`

The `Set Variant Variable to Null` method does not include arguments.

Example

The following example calculates the average of ten test score values. If any score is negative, then it sets this value to `Null`. Before this example calculates the average, the `IsNull` statement reduces the total count of scores to only those scores that are positive values:

```
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
```

Related Topics

- [Is Variable Null Method](#)
- [Is Variable Set Method](#)
- [Get Variant Type Method](#)

String Methods

This topic describes string methods. It includes the following topics:

- [Compare Strings Method](#)
- [Compare Strings Operator](#)
- [Convert Number to String Method](#)
- [Convert String to Lowercase Method](#)
- [Convert String to Uppercase Method](#)
- [Copy String Method](#)
- [Get a String of Spaces Method](#)
- [Get ANSI String Method](#)
- [Get First Number From String Method](#)
- [Get Left String Method](#)
- [Get Repeated Character String Method](#)
- [Get Right String Method](#)
- [Get String Length Method](#)
- [Get Substring Method](#)
- [Get Substring Position Method](#)
- [Remove Spaces From String Method](#)
- [Replace String Method](#)
- [Right-Justify String Method](#)

- [Set String Comparison Method](#)
- [Set String Format Method](#)
- [Trim Spaces From String Method](#)
- [Trim Trailing Spaces From String Method](#)

Compare Strings Method

The Compare Strings method compares two strings. It returns one of the following integers. This integer describes the result of the comparison:

- -1 (negative one). string1 is less than string2.
- 0 (zero). string1 equals string2.
- >1 (greater than one). string1 is greater than string2.
- Null . string1 equals Null or string2 equals Null.

This method passes the values in the *string1* argument and *string2* argument as a variant. It supports any type of expression. It automatically converts a number to a string.

Format

StrComp(*string1*, *string2*[, *compare*])

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string1</i>	An expression that contains the first string to compare.
<i>string2</i>	An expression that contains the second string to compare.
<i>compare</i>	An integer that specifies if the comparison is case-sensitive. You use one of the following values: 0. Case-sensitive. It performs a case-sensitive comparison according to the ANSI character set sequence. 1. Not case-sensitive. It performs a comparison that is not case-sensitive according to the relative order of characters. The country code setting for your computer determines this order. If you do not include the compare argument, then this method uses the module-level default. The Set String Comparison method sets this default. For more information, see Set String Comparison Method.

Example

The following example compares a custom string to the Smith string:

```
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
```

Compare Strings Operator

The Compare Strings operator is a standard Visual Basic operator that compares the contents of two strings. It returns one of the following values:

- -1 (TRUE). The *string* does match the *pattern*.
- 0 (FALSE). The *string* does not match the *pattern*.

If the *string argument* or if the *pattern argument* is NULL, then it returns NULL.

The Compare Strings operator performs a comparison according to the current configuration of the Set String Comparison method. For more information, see [Set String Comparison Method](#).

For more information about operators, see [About Expressions](#).

Format

string LIKE *pattern*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	Any string or string expression.
<i>pattern</i>	Any string expression to compare to the value of the <i>string argument</i> .

Usage

The following table describes characters that you can use in the *pattern argument*.

Character	Character That the Compare Strings Operator Compares
?	A single character.
*	A set of zero or more characters.
#	A single digit character in the range of 0 through 9.
[chars]	A single character in <i>chars</i> .

Character	Character That the Compare Strings Operator Compares
[!chars]	A single character not in <i>chars</i> .
[startchar–endchar]	A single character in the range <i>startchar</i> through <i>endchar</i> .
[!startchar–endchar]	A single character not in the range <i>startchar</i> through <i>endchar</i> .

A range or list can occur in a single set of square brackets. The Compare Strings operator matches ranges according to their ANSI values. In a range, *the value in startchar* must be less than the value in *endchar*.

Example

The following example determines if 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
```

Convert Number to String Method

The Convert Number to String method converts a number to a string. It returns a string representation of this number. It uses the following precision in the returned string:

- Single-precision for an integer or for a single-precision numeric expression.
- Double-precision for a long number or for a double-precision numeric expression.
- Currency precision for currency.
- A variant returns the precision of the underlying variable type. For more information, see [Variants](#).

Format

`Str[$] (number)`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	The number that this method converts to a string.

Argument	Description

Example

The following example prompts the user to enter two numbers. It adds these numbers, and then displays 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
```

Convert String to Lowercase Method

The Convert String to Lowercase method converts the contents of a string to lowercase, and then returns a lowercase copy of that string. It substitutes characters according to the country specified in the Microsoft Windows Control Panel. It accepts expressions of type string. It accepts any type of argument and converts the input value to a string.

If you must configure Siebel VB to compare text values but not case, then you can use the Convert String to Lowercase method and the Convert String to Uppercase method. For more information, see [Convert String to Uppercase Method](#).

Format

LCase[\$] (string)

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or an expression that contains a string. If this value is NULL, then this method returns a Null variant. For more information, see Variants .

Example

The following example converts to lowercase a string that the user enters:

```
Sub Button_Click
  Dim userstr as String
  userstr = "This Is A Test"
  userstr = LCase$(userstr)
End Sub
```

Convert String to Uppercase Method

The Convert String to Uppercase method converts lowercase letters to uppercase letters. It returns a copy of *this string*. Note the following:

- The conversion occurs according to the country specified in the Microsoft Windows Control Panel.
- It accepts any type of argument and converts the input value to a string.
- If the value that the *string argument contains* is NULL, then this method returns a Null variant. For more information, see [Variants](#).

Format

`UCase[$] (string)`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression.

Example

The following example converts a file name that the user enters to uppercase letters:

```
Option Base 1
Sub Button_Click
    Dim filename as String
    filename = "c:\temp\trace.txt"
    filename = UCase(filename)
End Sub
```

Copy String Method

The Copy String method copies one string to another string or assigns a custom variable to another variable. It does not return a value. It copies values differently depending on the following:

- The value in the *string argument* is shorter than *the value in the string-expression argument*. It copies the leftmost characters that *the string-expression contains* to the *string that the string argument identifies*. The number of characters it copies is equal to the length of the *string argument*.
- The value in the *string argument* is longer than *the value in the string-expression argument*. It copies every character in the *string-expression argument* to *the string that the string argument identifies*, filling it from left to right. It replaces any leftover characters in the *string argument* with spaces.

You cannot use the Copy String method to assign variables of different custom types if one of these variables contains a variant or a variable-length string.

Format A

```
Lset  
string = string-expression
```

Format B

```
Lset  
variable1 = variable2
```

If you use format B, then the number of characters that this method copies is equal to the length of the shorter of *variable1* and *variable2*.

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string variable or string expression to contain the copied characters.
<i>string-expression</i>	A string variable or string expression that contains the string that this method copies.
<i>variable1</i>	A variable in a custom type to contain the copied variable.
<i>variable2</i>	A variable that contains a custom type to be copied.

Example

The following example places a user last name into a variable. If the name is longer than the size of **lastname**, then it truncates the user name:

```
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
```

Get a String of Spaces Method

The Get a String of Spaces method returns a string of spaces.

Format

```
Space[$] (number)
```

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	A numeric expression that specifies the number of spaces to return. This number can be any numeric data type, but this method rounds the number to an integer. <i>This number</i> must reside in the range of 0 through 32,767.

Example

For an example, see [Get Octal Method](#).

Get ANSI String Method

The Get ANSI String method returns the character string that corresponds to the ANSI code of the character that the `charCode` argument contains.

Format

`Chr[$] (charCode)`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>charCode</i>	An integer in the range of 0 through 255 that identifies the ANSI code for a character.

Example

The following example displays the character equivalent for an ASCII code in the range of 65 through 122:

```
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
```

Get First Number From String Method

The Get First Number From String method returns the numeric value of the first number that it finds in a string. If it finds no number, then it returns 0 (zero). It ignores any spaces that exist in the string.

Format

`Val(string)`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression that contains a number.

Example

The following example examines the value of the profit variable. If profit contains a negative number, then it displays 0 (zero). The Sgn statement determines if 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
```

Get Left String Method

The Get Left String method returns a string copied from the beginning of another string. If the value in the *length* argument is greater than the length of the string that the *string* argument specifies, then it returns the entire string.

Format

`Left[$](string, length)`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or an expression that contains a string. This method returns a substring of the string that you specify. The substring begins at the first character of this string. If this argument contains a NULL value, then this method returns a Null variant. For more information, see Variants. This method accepts any type of <i>string</i>, including numeric values, and converts the input value to a string.
<i>length</i>	An integer that specifies the number of characters to copy.

Example

The following example gets a user first name from the entire name:

```
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
```

Get Repeated Character String Method

The Get Repeated Character String method creates a string that consists of a repeated character. It returns this string.

Format A

```
String[$] (number, character)
```

Format B

```
String[$] (number, stringExpression)
```

For information about the dollar sign, see [Usage of the Dollar Sign](#).

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	The length of the string that this method returns. <i>This number</i> must reside in the range of 0 through 32,767.
<i>character</i>	An integer or integer expression that contains the ANSI code of the character that this method uses. This argument must evaluate to an integer in the range of 0 through 255.

Argument	Description
<i>stringExpression</i>	A string argument. This method uses the first character of this argument as the repeated character.

Example

The following example places asterisks (*) in front of a string that it prints 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
```

Get Right String Method

The Get Right String method returns a portion of a string beginning at the end of the string. Note the following:

- It accepts any type of *string*, including numeric values, and converts the input value to a string.
- If the value that the *string argument contains* is NULL, then it returns a Null variant. For more information, see [Variants](#).
- If the *value in the length* argument is greater than the length of the *string*, then it returns the entire string.

Format

```
Right[$] (string  
    ,  
    length)
```

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression that contains the characters to copy.
<i>length</i>	The number of characters to copy, beginning at the right-most position of the string and counting toward the left.

Example

The following example searches for the BMP extension in a file name that the user enters. If it does not find the file, then it activates the Paintbrush application. It uses the Option Compare Text statement to accept uppercase or lowercase letters for the file name extension. For more information, see [Set String Comparison Method](#):

```
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
```

Get String Length Method

The Get String Length method gets the length of one of the following:

- The *string* that the *string* argument contains
- The string in the variable that the *varName* argument identifies

Format A

`Len(string)`

Format B

`Len(varName)`

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	Includes a string or an expression that evaluates to a string. If you use the string argument, then this method returns the number of characters that the string contains.
<i>varName</i>	Identifies a variable that contains a string. If the varName argument: • Identifies a variant variable. This method returns as a string the number of bytes required to identify the value of this variable. • Identifies a variant <i>variable</i> that contains NULL. This method returns a Null variant. • Does not identify a variant variable. This method returns the length of the predefined data type or the custom type.

Example

The following example returns the length of a name that the user enters, including spaces:

```
Sub Button_Click
  Dim username as String
  username = "Chris Smith"
  count = Len(username)
End Sub
```

Get Substring Method

The Get Substring method returns a portion of a string, starting at a location in the string that you specify. It allows you to get a substring from a string. It accepts any type of string, including numeric values, and converts the input value to a string. To modify a portion of a string, see [Replace String Method](#).

Format

```
Mid[$] (string
,
start[,
length])
```

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression that contains the string that this method copies. If this value is NULL, then this method returns a Null variant. Mid\$ requires the string argument to be of type string or variant. For more information, see Variants .
<i>start</i>	An integer that identifies the starting position in the <i>string argument</i> to begin copying characters. The position of the first character in a string is 1. If the number in the start argument is larger than the number of positions that the string contains, then this method returns a null string ("").
<i>length</i>	An integer that specifies the number of characters to copy.

Example

The following example returns the last name in a string that the user enters:

```
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
```

Get Substring Position Method

The Get Substring Position method returns the position of the first occurrence of a substring that resides in another string. It can also return the following values:

- Returns a zero in the following situations:
 - *The value in the start argument is greater than the length of the value in the string2 argument.*
 - *The string1 argument contains a null string.*
 - *The string2 argument is not found.*
- If the *string1 argument* or the *string2 argument contains* a null variant, then it returns a null variant.
- If *the string2 argument contains* a null string (""), then it returns the value of the *start argument*.

Format A

```
InStr([start
,]
string1
,
string2)
```

Format B

```
InStr(start, string1
,
string2[, compare])
```

Arguments

Arguments can be of any type. This method converts them to strings.

The following table describes the arguments that you can use with this method.

Argument	Description
<i>start</i>	An integer that identifies a position in the string that the <i>string1 argument identifies</i> . <i>This position is the starting position</i> , with the first character in the string as 1. If you do not specify a value for the start argument, then this method starts at the beginning of the string.
<i>string1</i>	The string that includes the substring.
<i>string2</i>	The substring.
<i>compare</i>	You can use one of the following values:

Argument	Description
	0. Perform a search that is case-sensitive according to the ANSI character set sequence. 1. Perform a search that is not case-sensitive according to the relative order of characters as determined by the country code setting for your computer. If you do not include the <i>compare argument</i>, then this method uses the module level default that the Set String Comparison method sets. For more information, see Set String Comparison Method.

Example

The following example creates a random string of characters, and then uses the Get Substring Position method to find the position of a single character in 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)
  Randomize
  For x = 1 to 26
    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
```

Remove Spaces From String Method

The Remove Spaces From String method removes leading spaces from a string. It returns a copy of this *string* with the leading spaces removed. It accepts any type of *string*, including numeric values, and converts the input value to a string.

If the value that the *string argument contains is NULL*, then this method returns a Null variant. For more information, see [Variants](#).

Format

LTrim[\$] (*string*)

For information about the dollar sign, see *Usage of the Dollar Sign*.

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression that contains the string that this method trims.

Example

The following example removes the leading spaces from a string:

```
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
```

Replace String Method

The Replace String method replaces part or all of one string with another string. It returns the string that the *stringVar* argument contains.

Format

Mid (*stringVar*
,
***start*[,**
***length*]) =**
string

The following table describes the arguments that you can use with this method.

Argument	Description
<i>stringVar</i>	The string that this method modifies.
<i>start</i>	An integer that identifies the position in <i>stringVar</i> at which to begin replacing characters. The position of the first character in a string is 1.

Argument	Description
<i>length</i>	An integer that identifies the number of characters to replace.
<i>string</i>	The string that this method places into <i>stringVar</i> .

Usage

This method replaces characters starting at the *start* position to the end of the *string* in the following situations:

- You do not include the *length* argument.
- The string that the *string* argument contains is shorter than the value that the *length* argument contains.

This method does the following:

- If the *start* value is larger than the number of characters in *stringVar*, then it appends the *string* that the *string* argument contains to *stringVar*.
- If *length* plus *start* is greater than the length of *stringVar*, then it replaces characters only up to the end of *stringVar*.
- If the *value* in the *start* argument is greater than the number of characters that exist in *stringVar*, then it creates an error at runtime.

The Replace String method never modifies the number of characters in the *string Var* argument.

Example

The following example replaces the last name in a custom string with asterisks (*):

```
Sub Button_Click
  Dim username as String
  Dim position as Integer
  Dim count as Integer
  Dim unname as String
  Dim replacement as String
  username = "Chris Smith"
  unname = 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(unname, count) = replacement
  Next x
End Sub
```

Right-Justify String Method

The Right-Justify String method right-justifies one string in another string. It does not return a value. It justifies a string in the following ways:

- Value in the *string argument* is longer than *string-expression*. It replaces the leftmost characters of *the string* with spaces.
- Value in the *string argument* is shorter than *string-expression*. It copies only the leftmost characters of *string-expression*.

You cannot use it to assign variables of different custom types.

Format

```
Rset  
string = string-expression
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	The string that receives the right-aligned characters.
<i>string-expression</i>	The string that this method puts into the <i>string that the string argument contains</i> .

Example

The following example right-justifies an amount that the user enters 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
```

Set String Comparison Method

The Set String Comparison method specifies the default method for string comparisons to case-sensitive or not case-sensitive. It does not return a value. You must include this method in the general declarations section. Note the following:

- A binary comparison is case-sensitive. It compares strings according to the ANSI character set. A lowercase letter is different from an uppercase letter.
- A text comparison is not case-sensitive. It compares strings according to the relative order of characters. The country code setting for your computer determines this order.

Format

```
Option Compare {Binary | Text}
```

This statement does not include arguments.

Example

The following example compares the following strings:

- Jane Smith
- jane smith

If Set String Comparison is Text, then these strings are the same. If Set String Comparison is Binary, then these strings are not the same. Binary is the default value. To examine this difference, you can run the example, comment out the Set String Comparison method, and then 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
```

Set String Format Method

The Set String Format method returns an *expression* in a *format* that you specify. For details about how to use this method, see [About Formatting Strings](#).

Format

```
Format[$](expression[, format])
```

For information about the dollar sign, see *Usage of the Dollar Sign*.

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	The value that this method formats. This argument can contain one of the following items: • Number • String • Variant This method formats the <i>value you enter</i> as a number, date, time, or string depending on the <i>format</i> argument. To format a string, it transfers one character at a time from the value you enter in the <i>expression argument</i> to the output string.
<i>format</i>	A string expression that identifies the format that this method uses. You must use quotation marks (""") to enclose the value in the <i>format</i> argument.

Usage

The `Set String Format` method formats a numeric value as a number or date and time. If you specify a numeric expression, and if you do not include the *format* argument, or if the *format* argument is null, then this method converts the number to a string without any special formatting.

It can format a numeric value or a variant as a date. If it formats a numeric value as a date, then it interprets the value according to standard Visual Basic date encoding. The starting date is December 30, 1899. It represents this date as zero. It represents subsequent dates as the number of days from the starting date.

If the *format* string does not match the Regional Settings, or if the Date in the Microsoft Windows setting is not set to the U.S. format, then the Set String Format method does not return the correct format.

Example 1

The following example includes tokens:

[illegible]

Example 2

The following example calculates the square root of 2 as a double-precision, floating point value, and then 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
```

Example 3

The following example uses multiple tokens to format the result of the Now method, which returns the current date and time on the computer 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, see the following topics:

- [Convert Expression to Currency Method](#)
- [Calculate Future Value Method](#)
- [Go To Statement](#)

Trim Spaces From String Method

The Trim Spaces From String method removes leading and trailing spaces from a string. It returns a copy of this string with the leading and trailing spaces removed. Note the following:

- It accepts expressions of type string.
- It accepts any type of *string*, including numeric values, and converts the input value to a string.
- If the value that the *string argument contains* is NULL, then this method returns a Null variant. For more information, see [Variants](#).

Format

`Trim[$] (string)`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A literal or expression from which this method removes leading and trailing spaces.

Example

For an example, see [Get Substring Method](#).

Trim Trailing Spaces From String Method

The Trim Trailing Spaces From String method copies a string, and then removes any trailing spaces that exist in that copy. It returns a string with all trailing spaces removed. Note the following:

- It accepts any type of *string*, including numeric values, and converts the input value to a string.
- If the value that the *string argument contains* is NULL, then it returns a Null variant. For more information, see [Variants](#).

Format

RTrim[\$] (*string*)

For information about the dollar sign, see [Usage of the Dollar Sign](#).

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string or string expression.

Example

For an example, see [Right-Justify String Method](#).

Array Methods

This topic describes array methods. It includes the following topics:

- [Declare Array Method](#)
- [Erase Array Method](#)
- [Get Array Lower Boundary Method](#)
- [Get Array Upper Boundary Method](#)
- [Set Array Lower Boundary Method](#)

For more information, see [Arrays](#).

Declare Array Method

The Declare Array method allocates memory for a dynamic array to the dimensions that you specify. It does not return a value. You typically use the Dim statement to declare a dynamic array without specifying a size for it.

Format

ReDim [Preserve] *arrayName* (*lower To upper*) [As [New] *type*], ...

The following table describes the arguments that you can use with this method.

Argument	Description
<i>arrayName</i>	The name of the array to redimension.
<i>lower</i>	The lower boundary of the array. If you do not specify this <i>argument</i> , then Siebel VB uses 0 as the default value of the lower boundary. You can use the Set Array Lower Boundary method to modify this default value.
<i>upper</i>	The upper boundary of the array.
<i>type</i>	The data type of the array elements.

Usage

Note the following:

- You can use the ReDim statement to declare an array in a procedure that has not previously been declared using the Dim statement or the Global statement. In this situation, you can declare no more than 60 dimensions.
- As an option, you can use the Declare Array method to reset array elements.
- You cannot use the Declare Array method at the module level. You must use it in a procedure.
- The Preserve option modifies the last dimension in the array while maintaining the contents of the array. If you do not specify the Preserve option, then Siebel VB resets the contents of the array. It sets numbers to zero (0). It sets strings and variants to null ("").
- If you do not include the As clause, then to specify the variable type, you can use a type character as a name suffix. You can use this clause and suffix in a single Declare Array method. You cannot use this clause and suffix on the same variable.
- You must not redimension an array in a procedure that has received a reference to an element in the array in an argument. The result of this configuration is not predictable.
- A dynamic array that you create with the Dim statement can include no more than eight dimensions. If you require more than eight dimensions, then you must use the Declare Array method. For information about declaring a dynamic array, see [Dynamic Arrays](#).

Example

The following example determines 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 this example redimensions 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
```

Related Topics

- [Declare Variable Statement](#)
- [Declare Global Variable Statement](#)

Erase Array Method

The Erase Array method erases the contents of a fixed array or frees the storage associated with a dynamic array. It does not return a value. The following table describes how it erases the contents of different element types.

Element Type	Description
Numeric	Sets each element to zero.
Variable-length string	Sets each element to a zero-length string ("").
Fixed-length string	Fills the string of each element with zeros.
Variant	Sets each element to Empty.
Custom type	Clears each member of each element as if each member is an array element. For example, it sets numeric members to zero, strings to "", and so on.
Object	Sets each element to the following value: Nothing

Format

```
Erase Array[, Array]
```

The following table describes the arguments that you can use with this method.

Argument	Description
Array	The name of the array to erase.

Argument	Description

Example

The following example prompts the user for a list of item numbers to put into an array. If the user must start over, then it clears the array. For information about the Dim statement, see *Declare Variable Statement*:

```
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
```

Get Array Lower Boundary Method

The Get Array Lower Boundary method returns the lowest index number of the dimension that the *dimension argument* identifies. The numbering for each dimension of an array starts at 1. If you do not include the *dimension argument*, then it uses 1 is the default.

You can use the Get Array Lower Boundary method with the Get Array Upper Boundary method to determine the length of an array.

Format

```
LBound(arrayname [, dimension] )
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>arrayname</i>	The name of the array to query.

Argument	Description
<i>dimension</i>	The dimension to query.

Example

The following example resizes an array if the user enters more data than this array can hold. It uses the LBound statement and the UBound statement to determine the existing size of the array. It uses the ReDim statement to resize the array. The Option Base statement sets the default lower boundary 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
```

Get Array Upper Boundary Method

The Get Array Upper Boundary method returns the upper boundary an array. You can use the Get Array Lower Boundary method with the Get Array Upper Boundary method to determine the length of an array.

The minimum value for the lower boundary of an array is 0. The maximum value for the upper boundary of an array is 65,536.

Format

```
UBound(arrayName[, dimension])
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>arrayName</i>	The name of the array to query.
<i>dimension</i>	The array dimension whose upper boundary this method returns. If you do not include this argument, <i>then Siebel VB uses 1 as the default value.</i>

Argument	Description

Example

For an example, see [Get Array Lower Boundary Method](#).

Set Array Lower Boundary Method

The Set Array Lower Boundary method specifies the default lower boundary to use for an array. It does not return a value. Note the following:

- You can use this method to set the lower boundary before you use any array in your code. If you do not use this method, then Siebel VB uses 0 as the default value for the lower boundary.
- You can use this method only one time for each module.
- You cannot use this method in a procedure.
- You must include this method in the general declarations section. For example:

```
Option Explicit
Option Base 1
Option Compare Text
```

Format

```
Option Base lowerBound
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>lowerBound</i>	The value 0 or 1 or an expression that evaluates to one of these values.

Example

For an example, see [Get Array Lower Boundary Method](#).

Mathematical Methods

This topic describes mathematical methods. It includes the following topics:

- [Convert Number to Date Method](#)
- [Get Absolute Value Method](#)
- [Get ANSI Integer Method](#)

- *Get Arctangent Method*
- *Get Cosine Method*
- *Get Hexadecimal Method*
- *Get Integer Method*
- *Get Rounded Integer Method*
- *Get Logarithm Method*
- *Get Octal Method*
- *Get Number Sign Method*
- *Get Random Number Method*
- *Get Sine Method*
- *Get Square Root Method*
- *Get Tangent Method*
- *Randomize Method*

Overview of Mathematical Methods

This topic describes an overview of mathematical methods.

How Some Math Methods Handle the Data Type

Some math methods provide a different return value depending on the following data types of the variable that contains the return value:

- Integer or currency. The return value is a single-precision number.
- A single-precision value. The return value is a single-precision number.
- Long or variant. The return value is a double-precision value.
- A double-precision value. The return value is a double-precision value.

These methods are noted in this chapter, where appropriate.

Trigonometric Methods

The following information lists the trigonometric methods that are available in Siebel VB.

Method	Computation
ArcCoSecant	$\text{ArcCoSec}(x) = \text{Atn}(x/\text{Sqr}(x*x-1)) + (\text{Sgn}(x)-1)*1.5708$
ArcCosine	$\text{ArcCos}(x) = \text{Atn}(-x/\text{Sqr}(-x*x+1)) + 1.5708$
ArcCoTangent	$\text{ArcTan}(x) = \text{Atn}(x) + 1.5708$

Method	Computation
ArcSecant	$\text{ArcSec}(x) = \text{Atn}(x/\text{Sqr}(x*x-1)) + \text{Sgn}(x-1)*1.5708$
ArcSine	$\text{ArcSin}(x) = \text{Atn}(x/\text{Sqr}(-x*x+1))$
CoSecant	$\text{CoSec}(x) = 1/\text{Sin}(x)$
CoTangent	$\text{CoTan}(x) = 1/\text{Tan}(x)$
Hyperbolic ArcCoSecant	$\text{HArcCoSec}(x) = \text{Log}((\text{Sgn}(x)*\text{Sqr}(x*x+1)+1)/x)$
Hyperbolic ArcCosine	$\text{HArcCos}(x) = \text{Log}(x+\text{Sqr}(x*x-1))$
Hyperbolic ArcCoTangent	$\text{HArcCoTan}(x) = \text{Log}((x+1)/(x-1))/2$
Hyperbolic ArcSecant	$\text{HArcSec}(x) = \text{Log}((\text{Sqr}(-x*x+1)+1)/x)$
Hyperbolic ArcSine	$\text{HArcSin}(x) = \text{Log}(x+\text{Sqr}(x*x+1))$
Hyperbolic ArcTangent	$\text{HArcTan}(x) = \text{Log}((1+x)/(1-x))/2$
Hyperbolic CoSecant	$\text{HCoSec}(x) = 2/(\text{Exp}(x)-\text{Exp}(-x))$
Hyperbolic Cosine	$\text{HCos}(x) = (\text{Exp}(x)+\text{Exp}(-x))/2$
Hyperbolic Cotangent	$\text{HCotan}(x) = (\text{Exp}(x)+\text{Exp}(-x))/(\text{Exp}(x)-\text{Exp}(-x))$
Hyperbolic Secant	$\text{HSec}(x) = 2/(\text{Exp}(x)+\text{Exp}(-x))$
Hyperbolic Sine	$\text{HSin}(x) = (\text{Exp}(x)-\text{Exp}(-x))/2$
Hyperbolic Tangent	$\text{HTan}(x) = (\text{Exp}(x)-\text{Exp}(-x))/(\text{Exp}(x)+\text{Exp}(-x))$
Secant	$\text{Sec}(x) = 1/\text{Cos}(x)$

Exponential Method

The Exponential method returns the value *e* raised to the power of the value that you specify in the *number argument*. The value *e* is the base of natural logarithms. The constant *e* is approximately 2.718282. For information about how this method handles the data type, see [How Some Math Methods Handle the Data Type](#).

Format

Exp (*number*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	The exponent value of e.

Example

The following example estimates the value of a factorial of a number that the user enters:

```
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
```

About the Factorial

A factorial 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. An exclamation point (!) can specify a factorial.

Get Absolute Value Method

The Get Absolute Value method returns the absolute value of the value that the *number argument contains*. The data type of the return value matches the data type of the value in the *number argument*. This method does the following:

- If *the value* is a string variant type, then it converts the return value to a double variant type.
- If the value is an empty variant type, then it converts the return value to a long variant type.

For more information, see [Variants](#).

Format

Abs (*number*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	Any valid numeric expression.

Example

The following example determines the difference between the oldacct variable and the newacct variable:

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

Get ANSI Integer Method

The Get ANSI Integer method returns an integer that corresponds to the ANSI code of the first character in the string that you specify. You can use the Get ANSI String method to modify an ANSI code to string characters. For more information, see [Get ANSI String Method](#).

Format

Asc(*string*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>string</i>	A string expression that contains one or more characters.

Example

The following example asks the user for a letter and returns the ANSI value for this letter:

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

Get Arctangent Method

The Get Arctangent method returns the angle in radians of the arctangent of a number that you specify. It assumes *the value in the number argument* is the ratio of two sides of a right triangle: the side opposite the angle to find and the

side adjacent to the angle. It 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 any of the following numeric expressions:

- Long
- Variant
- Double-precision

You can multiply a value by $(180/\text{PI})$ to convert radians to degrees. The value of PI is approximately 3.14159.

Format

Atn (*number*)

This method uses the same arguments as the Get Absolute Value method. For more information, see [Get Absolute Value Method](#).

Example

The following example finds the roof angle necessary for a house that includes an attic ceiling height of 8 feet at the roof peak and a 16 foot span from the outside wall to the center of the house. The Get Arctangent method returns the angle in radians. It multiplies this value by $180/\text{PI}$ 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
```

Get Cosine Method

The Get Cosine method returns the cosine of an angle. The angle can be positive or negative. The return value is between negative 1 and 1. To convert degrees to radians, this method multiplies the return value by $(\text{PI}/180)$. The value of PI is approximately 3.14159.

Format

Cos (*number*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	An angle in radians.

Example

The following example finds the length of a roof, given the roof pitch, and the distance of the house from the house center to the outside wall of the house:

```
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
```

Get Hexadecimal Method

The Get Hexadecimal Method returns the hexadecimal representation of the value in the *number argument*. It returns this value in a string. If the value in the *number argument*:

- Is an integer. The return string contains up to four hexadecimal digits.
- Is not an integer. This method converts the value to a long integer, and the string can contain up to eight hexadecimal digits.

You precede the hexadecimal value with the following characters to express a hexadecimal number:

&H

For example, &H10 equals decimal 16 in hexadecimal notation.

Format

Hex[\$] (number)

For information about the dollar sign, see [Usage of the Dollar Sign](#).

This method uses the same arguments as the Get Absolute Value method. For more information, see [Get Absolute Value Method](#).

Example

The following example returns the hex value for a number that the user enters:

```
Sub Button_Click
Dim usernum as Integer
Dim hexvalue as String
usernum = 23
hexvalue = Hex(usernum)
End Sub
```

Get Integer Method

The Get Integer method returns the integer part of a number:

- For a positive number, it removes the fractional part of the expression and returns the integer part only.

- For a negative number, it returns the largest integer that is less than or equal to the expression. For example:
- `Int(6.2)` returns 6.
- `Int(-6.2)` returns negative 7.

Format

`Int(number)`

This method uses the same arguments as the [Get Absolute Value Method](#). For more information, see [Get Absolute Value Method](#).

Similarities Between the Get Integer Method and the Get Rounded Integer Method

The `Get Integer` method performs the same work as the `Get Rounded Integer` method except it handles negative numbers differently. For example:

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

For more information, see [Get Rounded Integer Method](#).

How The Get Integer Method Handles Variant Types

The return type matches the type of the numeric expression. This includes a variant expression that returns the same variant type as the input value except for the following items:

- The string variant type returns as the double variant type.
- An empty variant type returns as a long variant type.

For more information, see [Variants](#).

Example

The following example uses the `Get Integer` method to create random numbers in the range of ASCII values for lowercase a through z. This ASCII range is 97 through 122. It converts the values to letters and displays them 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)
  Randomize
  For x = 1 to 26
    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
```

Get Rounded Integer Method

The Get Rounded Integer method removes the fractional part of a number. It returns the integer part of a number. For positive and negative numbers, it removes the fractional part of the expression and returns the integer part only. For example:

- Fix (6.2) returns 6.
- Fix (-6.2) returns negative 6.

For more information, see [Get Integer Method](#).

Format

Fix (number)

This method uses the same arguments as the Get Absolute Value method. For more information, see [Get Absolute Value Method](#).

How The Get Integer Method Handles Variant Types

The data type of the return value matches the data type of the numeric expression. This includes variant expressions unless the numeric expression is one of the following:

- A string variant type that evaluates to a number. In this situation, the data type of the return value is a double variant type.
- An empty variant type. In this situation, the data type of the return value is a long variant type.

For more information, see [Variants](#).

Example

The following example returns the integer portion of a number that the user enters:

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

Get Logarithm Method

The Get Logarithm method returns the natural logarithm of a number. For information on how this method handles the data type, see [How Some Math Methods Handle the Data Type](#).

Format

Log (number)

This method uses the same arguments as the Get Absolute Value method. For more information, see [Get Absolute Value Method](#).

Example

The following example uses the Get Logarithm method to determine which of the following numbers is larger:

- 999^{1000} (999 to the 1000th power)
- 1000^{999} (1000 to the 999th power):

```
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
```

You cannot use the exponent (^) operator for very large numbers, such as 999^{1000} .

Get Octal Method

The Get Octal method converts a number to an octal (base 8) number. It returns the octal representation of a number as a string. The following data type of the integer determines the octal digits that the string can contain:

- Is an integer data type. The string can contain up to six octal digits.
- Is not an integer data type. This method converts the expression to a long data type. The string can contain up to 11 octal digits.

To represent an octal number, you precede the octal value with the following code:

&O

For example, the following code equals decimal 8 in octal notation:

&O10

Format

`Oct[$] (number)`

For information about the dollar sign, see *Usage of the Dollar Sign*.

This method uses the same arguments as the Get Absolute Value method. For more information, see *Get Absolute Value Method*.

Example

The following example prints the octal values for the numbers 1 through 15:

```
Sub Button_Click
  Dim x As Integer, y As Integer
  Dim msgtext As String
  Dim nofspace As Integer
  msgtext = "Octal numbers from 1 to 15:" & Chr(10)
  For x = 1 to 15
    nofspace = 10
    y = Oct(x)
    If Len(x) = 2 then
      nofspace = nofspace - 2
    End If
    msgtext = msgtext & Chr(10) & x & Space(nospace) & y
  Next x
End Sub
```

Get Number Sign Method

The Get Number Sign method returns a value that identifies the sign of a number. It returns a different value depending on the following value of the number:

- Number is less than zero. It returns negative 1.
- Number is equal to zero. It returns 0.
- Number is greater than zero. It returns 1.

Format

Sgn (*number*)

This method uses the same arguments as the Get Absolute Value method. For more information, see [Get Absolute Value Method](#).

Example

The following example examines the value of the profit variable. If this value is a negative number, then it displays 0 for profit:

```
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
```

Get Random Number Method

The Get Random Number method returns a single-precision, pseudo-random number between 0 and 1 depending on the following value that you specify in the number argument:

- Less than zero. Uses the number you specify as the seed for a pseudo-random number. It creates this number every time the Get Random Number method runs.
- Equal to zero. Uses the number most recently created.
- Greater than zero or no value. Creates a sequence of pseudo-random numbers. Each run of the Get Random Number method uses the next number in the sequence. It creates the same sequence of random numbers when it runs unless the Randomize method resets the random number generator.

Because this method always uses the same algorithm and the same seed value to create a number, it creates a pseudo-random number rather than a true random number. The number it creates appears to be random unless it runs a very large number of iterations, at which point a pattern of numbers might emerge. For practical purposes, you can consider this pseudo-random number to fulfill the requirements you might have to create a random number.

Format

Rnd[(*number*)]

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	A numeric expression that describes how to create the random number.

Example

For an example, see [Randomize Method](#).

Get Sine Method

The Get Sine method returns the sine of an angle specified in radians. The return value is between -1 and 1. You specify the angle in radians as a positive or negative number. To convert degrees to radians, this method multiplies degrees by (PI/180). The value of PI is 3.14159.

For information on how this method handles the data type, see [How Some Math Methods Handle the Data Type](#).

Format

Sin(*number*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	A numeric expression that contains a number that represents the size of an angle in radians.

Example

The following 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
```

Get Square Root Method

The Get Square Root method returns the square root of a number. For information on how this method handles the data type, see [How Some Math Methods Handle the Data Type](#).

Format

Sqr (number)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	An expression that contains the number whose square root is to be found.

Example

For an example that calculates the square root of 2 as a double-precision floating-point value and displays it in scientific notation, see [Set String Format Method](#).

Get Tangent Method

The Get Tangent method returns the tangent of an angle in radians. You specify the value in the number argument in radians. This value can be positive or negative. To convert degrees to radians, this method multiplies degrees by PI/180. The value of PI is 3.14159.

For information on how this method handles the data type, see [How Some Math Methods Handle the Data Type](#).

Format

Tan (*number*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	A numeric expression that contains the number of radians in the angle whose tangent this method returns.

Example

The following example finds the height of the exterior wall of a building, given the 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
```

Randomize Method

The Randomize method creates a starting value for the random number generator. It does not return a value. If you do not specify a value for the *number* argument, then it uses the Get Seconds method to reset the random number generator.

Format

Randomize [*number*]

The following table describes the arguments that you can use with this method.

Argument	Description
<i>number</i>	An integer value that is in the range of negative 32768 through 32767.

Example

The following example uses the Randomize method and the Get Random Number method to create a random string of characters. The second For Next loop slows down processing in the first For Next loop so that the Timer method can seed the Randomize method with a new value each time the loop runs:

```
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)
  Randomize
  For x = 1 to 26
    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
```

Date and Time Methods

This topic describes date and time methods. It includes the following topics:

- *Convert Number to Date Method*
- *Convert Serial Number to Date Method*
- *Convert String to Date Method*
- *Convert String to Time Method*
- *Extract Day From Date-Time Value Method*
- *Extract Hour From Date-Time Value Method*
- *Extract Minute From Date-Time Value Method*
- *Extract Month From Date-Time Value Method*
- *Extract Second From Date-Time Value Method*
- *Extract Weekday From Date-Time Value Method*
- *Extract Year From Date-Time Value Method*
- *Get Current Date Method*
- *Get Current Date and Time Method*
- *Get Current Time Method*
- *Get Current Seconds Method*
- *Get Serial Time Method*
- *Set Date Method*
- *Set Time Method*

Convert Number to Date Method

The Convert Number to Date method converts an expression to the data type variant of type date. Note the following:

- It returns a date variable type that represents a date from January 1, 100, through December 31, 9999. A value of 2 represents January 1, 1900.
- It represents a time as fractional days.
- It accepts string and numeric values.
- It converts the time portion of a date expression in the following situations:
 - If you include a time portion as part of the expression.
 - If the time expression is the only argument.

To compare dates, you must make sure that you format these dates consistently with each other. You can use this method to convert both expressions before Siebel VB compares them.

For ways to display the desired result of a date conversion, see [Set String Format Method](#).

For more information, see [Variants](#).

Format

CVDate (*expression*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	Any expression that Siebel VB can evaluate to a number.

How the Operating System Affects the Date Format

The date format depends on the format that the operating system uses. For example, if the operating system uses the mm/dd/yyyy format, then the input argument must use the mm/dd/yyyy format or the mm-dd-yyyy format. If you use an integer value, then to calculate this date Siebel eScript adds this integer as the number of days after the year 1900. In this situation, it returns a date in the mm/dd/yyyy form.

Example

The following example displays the date for one week from a date that the user enters:

```
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
```

Convert Serial Number to Date Method

The Convert Serial Number to Date method converts a number to a date. It returns a date variable type that represents a date from January 1, 100, through December 31, 9999. For more information, see [Variants](#).

To specify a relative date, you can use a numeric expression for any of the arguments. This expression evaluates to a number of days, months, or years before or after a specific date.

Format

```
DateSerial(year, month, day)
```

The following table describes the arguments that you can use with this method. As an option, you can use a numeric expression instead of an integer for each argument.

Argument	Description
<i>year</i>	An integer that identifies a year in the range of 100 through 9999.
<i>month</i>	An integer that identifies a month in the range of 1 through 12.
<i>day</i>	An integer that identifies a day in the range of 1 through 31.

Example

The following 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
```

Convert String to Date Method

The Convert String to Date method converts a string to a date. It returns a date variable type that represents a date from January 1, 100, through December 31, 9999, where January 1, 1900, is 2. For more information, see [Variants](#). Note the following:

- It accepts multiple string representations for a date.

- It uses the international settings of the operating system to resolve a numeric date.
- Because this method handles dates and not times, if it receives time information then it modifies this time to 12:00:00 AM, which is midnight of the date value.

For ways to display the desired result of a date conversion, see [Set String Format Method](#).

Format

DateValue(*date*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>date</i>	Any numeric or string expression that can evaluate to a date and time or date value. <i>This value</i> can be of any type, including string.

If the value that the date argument contains is not in a valid date format, then this method returns a Type Mismatch error.

Using the DateValue Statement With the GetFormattedFieldValue Statement

To avoid a locale conflict, it is recommended that you do not use the DateValue statement together with the GetFormattedFieldValue statement. Note the following:

- The GetFormattedFieldValue statement formats a date according to the locale setting of the object manager.
- The DateValue statement expects a date string in a format according to the locale setting of the operating system.
- A Siebel Server can run multiple object managers with different language settings. For example, ENU, DEU, or FRA. However, the server operating system can only have one value selected for Regional Options, such as English (United States).

In this example, if a client connects to a DEU object manager on a Siebel Server where the Regional Options parameter is set to English (United States), then the GetFormattedFieldValue statement returns a string that contains a date that it formats for DEU, but the DateValue statement expects a string formatted for ENU. This situation results in the following error in Siebel VB:

Illegal Function Call

Example

The following example displays the date for one week from the date that the user enters:

```
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 = CDate(str1)
    nextweek = DateValue(str1) + 7
    msgtext = "One week from your date is: "
    msgtext = msgtext & Format(nextweek, "dddddd")
  Else
```

```
msgtext = "Invalid date or format. Try again."  
Goto i  
End If  
End Sub
```

Convert String to Time Method

The Convert String to Time method converts a string to time. It returns a variant of date variable type that represents a time in one of the following ranges:

- The time 0:00:00 through 23:59:59
- The time 12:00:00 A.M. through 11:59:59 P.M.

For more information, see [Variants](#).

Format

`TimeValue(time)`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>time</i>	Any numeric or string expression that can evaluate to a date and time or time value.

Example

The following example writes a variable to a file according to a comparison of the last saved time and the current time. It dimensions the variables that it uses for the Convert String to Time method as double. It does this to make sure the calculations work correctly according to their values:

```
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
```

Extract Day From Date-Time Value Method

The Extract Day From Date-Time Value method returns the day component of a date and time value. The return value is an integer variable type in the range of 1 through 31. If the return value is null, then it returns a null variable type. For more information, see [Variants](#).

Format

Day (*date*)

The argument that this method uses is the same as the argument that the Convert String to Date method uses. For more information, see [Convert String to Date Method](#).

Example

The following example finds the month in the range of 1 through 12 and the day in the range of 1 through 31 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
```

Extract Hour From Date-Time Value Method

The Extract Hour From Date-Time Value method returns the hour component of a date and time value. The return value is an integer variable type in the range of 0 through 23. Note the following:

- If the return value is null, then this method returns a null variable type. For more information, see [Variants](#).
- *The value in the time argument* can be of any type, including string.

- The value in the time argument is a double-precision value:
- The numbers to the left of the decimal point denote the date.
- The decimal value denotes the time from 0 to 0.99999.

You can use the Convert String to Time method to get the correct value for a specific time.

Format

Hour (time)

The argument that this method uses is the same as the argument that the Convert String to Time method uses. For more information, see [Convert String to Time Method](#).

If the value that the minute argument contains does not evaluate to a date and time or to a time value, then this method returns 0 (zero). For example:

- The value 13:26 or the 1:45:12 PM returns a valid result.
- The value 1326 returns a 0.

Extract Minute From Date-Time Value Method

The Extract Minute From Date-Time Value method returns the minute component of a date and time value. The return value is an integer variable type in the range of 0 through 59. If the return value is null, then this method returns a null variable type. For more information, see [Variants](#).

Format

Minute (time)

The argument that this method uses is the same as the argument that the Convert String to Time method uses. For more information, see [Convert String to Time Method](#).

If the value that the time argument contains does not evaluate to a date and time or to a time value, then this method returns 0 (zero). For example:

- The value 13:26 or the value 1:45:12 PM returns a valid results.
- The value 1326 returns a 0.

Example

The following example gets the hour, minute, and second of the last modification date and time of a file:

```
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
```

Extract Month From Date-Time Value Method

The Extract Month From Date-Time Value method returns the month component of a date and time value. The return value is an integer variable type in the range of 1 through 12. If the return value is null, then this method returns a null variable type. For more information, see [Variants](#).

Format

Month (*date*)

The argument that this method uses is the same as the argument that the Convert String to Date method uses. For more information, see [Convert String to Date Method](#).

If the value in the date argument does not evaluate to a date and time or to a time value, then this method returns 0 (zero). For example:

- The value 11/20 or the value 11-20-2001 returns a valid results.
- The value 1120 returns a 0.

Example

The following example finds the month in the range of 1 through 12 and the day in the range of 1 through 31 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
```

Extract Second From Date-Time Value Method

The Extract Second From Date-Time Value method returns the seconds component of a date and time value. The return value is an integer variable type in the range of 0 through 59. If the return value is null, then this method returns a null variable type. For more information, see [Variants](#).

Format

Second (*time*)

The argument that this method uses is the same as the argument that the [Convert String to Time Method](#) uses. For more information, see [Convert String to Time Method](#).

If the value in the time argument does not evaluate to a date and time or to a time value, then this method returns 0 (zero). For example:

- The value 13:26:39 or the value 1:45:12 PM returns a valid results.
- The value 1326 returns a 0.

Example

For an example, see [Extract Minute From Date-Time Value Method](#).

Extract Weekday From Date-Time Value Method

The [Extract Weekday From Date-Time Value](#) method returns the day of the week of a date and time value. It returns this value as an integer in the range of 1 through 7, where 1 is Sunday and 7 is Saturday.

It accepts any expression, including strings, and attempts to convert the input value to a date value.

The return value is an integer variable type. If the return value is null, then this method returns a null variable type. For more information, see [Variants](#).

Format

Weekday (date)

The argument that this method uses is the same as the argument that the [Convert String to Date Method](#) uses. For more information, see [Convert String to Date Method](#).

Example

The following example determines the day of the week when 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
```

Extract Year From Date-Time Value Method

The Extract Year From Date-Time Value method returns the year component of a date and time value. It returns this value as an integer in the range of 100 through 9999. It accepts any type of *date*, including strings, and attempts to convert the input value to a date value.

The return value is an integer variable type. If the return value is null, then this method returns a null variable type. For more information, see *Variants*.

Format

Year (*date*)

The argument that this method uses is the same as the argument that the Convert String to Date method uses. For more information, see *Convert String to Date Method*.

Example

The following example returns the year for today:

```
Sub Button_Click
  Dim nowyear
  nowyear = Year(Now)
End Sub
```

Get Current Date Method

The Get Current Date method returns a ten character string that includes the current date as determined by the computer clock.

Format

Date[\$]

For information about the dollar sign, see *Usage of the Dollar Sign*.

This method does not include arguments.

Example

The following example displays the date for one week from the current computer date:

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

Get Current Date and Time Method

The Get Current Date and Time method returns the current date and time as a date variable type according to the setting of the computer date and time. For more information, see [Variants](#).

You can use the Set String Format method to specify the format that Siebel CRM uses to display the date and time.

Format

`Now()`

This method does not include arguments.

Example

The following example finds the month in the range of 1 through 12 and the day in the range of 1 through 31 for this Thursday. For another example, see [Set String Format Method](#):

```
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
```

Get Current Time Method

This method returns a string that contains the current time. It returns an eight character string of the following format:

hh:mm:ss

where:

- *hh* is the hour
- *mm* is the minute
- *ss* is the second

The hour uses a 24 hour clock in the range of 0 through 23.

Format

`Time[$]`

For information about the dollar sign, see [Usage of the Dollar Sign](#).

This method does not include arguments.

Example

The following example writes data to a file if it has not been saved in 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
```

Get Current Seconds Method

The Get Current Seconds method returns the number of seconds that have elapsed since midnight. You can use the Get Current Seconds method and the Randomize statement to seed the random number generator. For more information, see [Randomize Method](#).

Format

Timer

This method does not include arguments.

Example

The following example uses the Get Current Seconds method 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
```

Get Serial Time Method

The Get Serial Time method returns a time as a variant of type 7 (date and time) for a specific hour, minute, and second.

To specify a relative time for each argument, you can use a numeric expression that identifies the number of hours, minutes, or seconds before or after a specific time.

Format

TimeSerial(*hour*, *minute*, *second*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>hour</i>	A numeric expression that contains a value in the range of 0 through 23 that identifies an hour.
<i>minute</i>	A numeric expression that contains a value in the range of 0 through 59 that identifies a minute.
<i>second</i>	A numeric expression that contains a value in the range of 0 through 59 that identifies a second.

Example

The following example displays the current time:

```
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
```

Set Date Method

The Set Date method sets the computer date. It does not return a value. Note the following:

- If you do not include the dollar sign (\$), *then the expression* can be a string that contains a valid date, a date variable type, or a string variable type. For more information, see [Variants](#).
- If the *expression argument* is not already a date variable type, then it attempts to convert this value to a valid date from January 1, 1980, through December 31, 2099.
- To identify the day, month, and year if a string contains three numbers that are separated by valid date separators, it uses the Short Date format in the International section of Microsoft Windows Control Panel. It recognizes month names in full or abbreviated form.

Format

`Date[$] = expression`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	A string in one of the following forms: • <i>mm-dd-yy</i> • <i>mm-dd-yyyy</i> • <i>mm/dd/yy</i> • <i>mm/dd/yyyy</i>

The following table describes the value you must specify for each item in the expression.

Value	Value You Must Specify
<i>mm</i>	A month expressed as a two-digit number in the range of 01 through 12.

Value	Value You Must Specify
<i>dd</i>	A day expressed as a two-digit number in the range of 01 through 31.
<i>yy</i>	A year expressed as a two-digit number in the range of 00 through 99.
<i>yyyy</i>	A year expressed as a four-digit number in the range of 1980 through 2099.

Example

The following example modifies the computer date to a date that the user enters:

```
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
```

Set Time Method

The Set Time method sets the computer time. It does not return a value.

If *the value in the expression argument* is not already a date variable type, then this method attempts to convert it to a valid time. It recognizes the time separator characters defined in the International section of the Microsoft Windows Control Panel. For more information, see [Variants](#).

Format

Time[\$] = *expression*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	An expression that evaluates to a valid time.

Usage of the Dollar Sign

If you include the dollar sign (\$), then the following items apply:

- The value in the expression argument must evaluate to a string that uses 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.
- It uses a 24 hour clock. For example, 6:00 P.M. is 18:00:00.

If you do not include the dollar sign (\$), then the following items apply:

- The expression argument can include a string that contains a valid date, or a date variable type of 8 (string).
- It accepts 12 hour and 24 hour clocks.

Example

The following example modifies the time of the computer 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
```

ODBC Methods

This topic describes methods that you can use with ODBC (Open Database Connectivity). It includes the following topics:

- [Overview of ODBC Methods](#)
- [ODBC Close Connection Method](#)
- [ODBC Get Errors Method](#)
- [ODBC Get Query Results Method](#)
- [ODBC Get Schema Method](#)
- [ODBC Open Connection Method](#)
- [ODBC Run Query Method](#)

- [ODBC Run Query and Get Results Method](#)
- [ODBC Save Results to File Method](#)

Overview of ODBC Methods

ODBC methods in Siebel VB only support non-Unicode databases.

You must not use some methods to query a Siebel database. Instead, use the Siebel Object Interfaces to query data in a Siebel database. Use an ODBC method only to query non-Siebel data.

ODBC Close Connection Method

The ODBC Close Connection method is a Siebel VB method that disconnects from an ODBC data source connection that the ODBC Open method established. It returns a variant that includes one of the following values:

- 0. Successful disconnect.
- -1. Connection is not valid.

If you call the ODBC Close Connection method with an argument that is not valid, then it replies with the undocumented return code of -2 (negative two), which indicates a data source connection that is not valid. The following items are examples of arguments that are not valid:

- An SQLClose(0) argument
- A variable argument that does not contain an initial value

Format

`SQLClose(connection)`

The following table describes the arguments that you can use with this method.

Argument	Description
connection	A named argument that the ODBC Open method returns. It must be a long integer. For information about named arguments, see Comments and Call Subroutine Method .

Example

The following example opens the data source named **SblTest**, gets the names in the ODBC data sources, and then 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
```

ODBC Get Errors Method

The ODBC Get Errors method is a Siebel VB method that gets detailed information about errors that occur during an ODBC method call. It returns errors for the last ODBC method and the last connection.

Format

`SQLError(destination())`

The following table describes the arguments that you can use with this method.

Argument	Description
<i>destination</i>	A two-dimensional array of type variant, where each row contains one error.

Usage

The ODBC Get Errors method returns detailed information for each detected error to the caller in the destination array. It fills each row of the destination array with information for one error. The following table describes how it fills elements of each row with data.

Element	Description
1	A character string that identifies the ODBC error class and subclass. The ODBC Get Errors method might return information for more than one error in the destination array. A 0 (zero) in the first element of a row identifies the end of error information in the destination array.
2	A numeric value that identifies the native error code of the data source.
3	A text message that describes the error.

If no errors from a previous ODBC method call are present, then it returns a 0 (zero) in the array at the first element of the first row. If the array is not two dimensional or if it does not provide for the return of the preceding three elements, then it returns an error message in the array at the first element of the first row.

Example

The following example forces an error as a way to test the ODBC Get Errors method:

```
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 choose 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
```

ODBC Get Query Results Method

The ODBC Get Query Results method is a Siebel VB method that gets the results of a pending query. It returns a variant that contains one of the following values:

- The number of rows in the result set or the maxRows requested. Indicates success .
- -1. Unable to get results, or no results pending .
- 0. The query did not find any data.

It uses a default value of 0 for any argument that you do not include. This is true for all arguments except for the fetchFirst argument.

Format

```
SQLRetrieve(connection, destination()[, maxColumns][, maxRows]
[, columnNames][, rowNumbers][, fetchFirst])
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>connection</i>	The long integer that the ODBC Open Connection method returns.
<i>destination</i>	A two-dimensional array of type variant. The first index of the array cannot exceed 100.
<i>maxColumns</i>	The number of columns that this method gets in the request. If this number specifies fewer columns than the result contains, then it discards the rightmost result columns until the result fits the size that you specify.
<i>maxRows</i>	The number of rows that this method gets in the request.

Argument	Description
<i>columnNames</i>	An integer. If the <i>columnNames</i> argument is not zero, then this method does the following work: • Sets the first row of the array to the column names according to how the database schema specifies them. • Returns the column names as the first row of the array <i>that you identify in the ref argument</i>.
<i>rowNumbers</i>	An integer. If the <i>rowNumbers</i> argument is not zero, then that this method returns the row names as the first column of the array <i>that you identify in the ref argument</i> . It clears the user array before it gets the results.
<i>fetchFirst</i>	A positive integer that causes the result set that this method repositions to the first row of the database. If the value in the <i>fetchFirst argument</i> is not zero, then it repositions to the first row. If the database does not support repositioning, then it returns a -1 (negative one) error is returned.

Usage

If you do not include the *maxColumns argument* or the *maxRows argument*, then the ODBC Get Query Results method does the following:

- To determine the maximum number of columns and rows to get, it uses the size of the array that you specify in the destination argument.
- Attempts to return the entire result set.

To get extra rows, you can set the *fetchFirst* argument to 0 and use the ODBC Get Query Results method again.

Using the ODBC Get Query Results Multiple Times

In some situations, you can use the ODBC Get Query Results method multiple times until the return value is 0. For example, if the result set includes one of the following items:

- More rows than the *array can contain*
- More rows than the value specified in the *maxRows argument*

Example

The following example gets 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)
' Run 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
```

ODBC Get Schema Method

The ODBC Get Schema method is a Siebel VB method that returns the following information:

- Data sources available
- Current user ID
- Names of tables names
- Types of table columns
- Other data source and database information

Format

```
SQLGetSchema connection, action, qualifier, ref()
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>connection</i>	A long integer that the ODBC Open method returns.
<i>action</i>	An integer that specifies what information to return. Some database products and some ODBC drivers might not support every action.
<i>qualifier</i>	A string.
<i>ref</i>	An array of type variant that stores the results. It must be an array even if it includes only one dimension with one element. You must make sure the destination array is properly dimensioned to support the action. If you do not do this, then this method returns an error.

Values You Can Use In the Action Argument

ODBC Get Schema Method describes the values you can use in the action argument. If the ODBC Get Schema method cannot find the requested information or if the connection is not valid, then it returns a -1 (negative one).

Value	Description
1	Get a list of available data sources. The dimension of <i>ref()</i> is 1.
2	Get a list of databases on the current connection. Siebel VB does not support this value.
3	Get a list of owners in a database on the current connection. Siebel VB does not support this value.
4	Get a list of tables on the specified connection. It returns every table. You cannot use the qualifier argument with <i>value 4</i> .
5	Get a list of columns in the table that the <i>qualifier argument and the ref argument</i> specifies. This table must include two dimensions. This method returns the column name and the ODBC data type.
6	Get the user ID of the current connection user.
7	Get the name of the current database.
8	Get the name of the data source of the current connection.
9	Get the name of the RDBMS that the data source uses. For example, DB2.
10	Get the server name of the data source.
11	Get the terminology that the data source uses to reference owners.
12	Get the terminology that the data source uses to reference a table.
13	Get the terminology that the data source uses to reference a qualifier.
14	Get the terminology that the data source uses to reference a procedure.

Example

The following example opens the data source named **SblTest**, gets the names in the ODBC data sources, and then 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

```

ODBC Open Connection Method

The ODBC Open Connection method is a Siebel VB method that establishes a connection to an ODBC data source. It returns a long integer that identifies a unique connection ID that you can use with other ODBC methods. If you include the *outputString* argument, then it returns the completed connection string in the *outputString* argument.

If it cannot establish a connection, then it returns an ODBC error with a negative numeric value. You can test for this value. For more information, see [ODBC Get Errors Method](#).

Format

```
SQLOpen(connectString, [outputString][, prompt])
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>connectString</i>	A string or string variable that includes the following information: - Data source name - User ID - Password - Any other information that the driver requires to make a connection with the data source The connectString argument typically uses the following format: "DSN=dataSourceName;UID=loginID;PWD=password" However, it must use the format that the ODBC driver requires. Some parts of this string might not be required. For more information on the string you use to access a Siebel application, see Siebel Technical Note #206 on My Oracle Support.
<i>outputString</i>	A string variable that holds the completed connection string.

Argument	Description
<i>prompt</i>	One of the following integers that specifies when to display the driver dialog box: • 1. Always display the driver dialog box. • 2. Display the driver dialog box only when the specification is not sufficient to make the connection. • 3. Display the driver dialog box only when the specification is not sufficient to make the connection. Dialog boxes that are not required are not available and cannot be modified. • 4. Do not display the driver dialog box. If the connection is not successful, then return an error. If you do not include the prompt argument, then Siebel VB uses value 2.

Example

The following example opens the data source named **SblTest**, gets the names in the ODBC data sources, and then 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
```

ODBC Run Query Method

The ODBC Run Query method is a Siebel VB method that runs an SQL statement on a connection that the ODBC Open method establishes. It returns the number of columns in the result set for SQL SELECT statements as a variant. The value that it returns depends which of the following statements you use:

- You use the SELECT statement with UPDATE, INSERT, or DELETE. It returns the number of rows that the statement affected.
- You use any other SQL statement. It returns 0.

If it cannot run the query on the data source, or if the connection is not valid, then it returns a negative error code.

If there are any pending results on the connection, then it replaces the pending results with the new results.

Format

SQLExecQuery(*connection*, *query*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>connection</i>	A long integer that the ODBC Open method returns.
<i>query</i>	A string that contains a valid SQL statement.

Example

The following 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)

' Run 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
```

ODBC Run Query and Get Results Method

The ODBC Run Query And Get Results method is a Siebel VB method that opens a connection to a data source, runs an SQL statement, returns the result, and then closes the connection. It returns a variant that includes one of the following values:

- Positive number. The request is successful. The number identifies the number of results that this method returned or the number of rows affected. Other SQL statements return 0.
- Negative error code. An error occurred. It cannot complete the connection, the query is not valid, or another error condition occurred.

Format

```
SQLRequest(connectString, query, outputString[], prompt[], columnNames[], ref())
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>connectString</i>	A string or string variable that specifies the data source. For more information on the connect string, see <i>ODBC Open Connection Method</i> .
<i>query</i>	An SQL query statement that this method runs.
<i>outputString</i>	A string variable that holds the completed connection string.
<i>prompt</i>	An integer that specifies when to display the driver dialog box. For more information about using the prompt argument, see <i>ODBC Open Connection Method</i> .
<i>columnNames</i>	An integer. If <i>columnNames</i> is not zero, then it returns column names in the first row of the array that the <i>ref</i> argument identifies. If you do not include the <i>columnNames</i> argument, then the default value is 0.
<i>ref</i>	An array of type variant that holds the results of the action you request. It must be an array even if it includes only one dimension with one element.

Example

The following example uses the ODBC Run Query And Get Results method:

```
Function WebApplet_PreInvokeMethod (MethodName As String) As Integer
    If MethodName = "queryExtSys" Then

        ' The following opens the datasource SVBTESTW and
        ' runs 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, run 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
```

ODBC Save Results to File Method

This Siebel VB method gets the results of a query and stores them in a file. It returns one of the following values:

- Successful. It returns a variant that contains the number of rows that exist in the result set.
- Not successful. It returns -1 (negative one).

The arguments must be named arguments. For information about named arguments, see [Comments](#) and [Call Subroutine Method](#).

Format

```
SQLRetrieveToFile(connection, destination[, columnNames][, columnDelimiter])
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>connection</i>	The long integer that the ODBC Open Connection method returns.
<i>destination</i>	A string or string variable that contains the file name and path to this file. This method stores the results in this file.
<i>columnNames</i>	This argument can include one of the following values: • Not zero. The first row contains the column headers according to database schema requirements. • 0. Does not get the column headers. The default value is 0.
<i>columnDelimiter</i>	The string this method uses to separate each field in a row. If you do not include the <i>columnDelimiter</i> argument, then it uses a tab character to separate each field.

Example

The following example opens a connection to a data source and saves 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)

' Run 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
```

Object Querying Methods

This topic describes object querying methods. It includes the following topics:

- [Compare Object Expressions Operator](#)
- [Is Expression a Date Method](#)
- [Is Object Of Class Method](#)
- [Is Optional Argument Missing Method](#)
- [Is Variable Null Method](#)
- [Is Variable Numeric Method](#)
- [Is Variable Set Method](#)

Compare Object Expressions Operator

The Compare Object Expressions operator compares two object expressions. It returns one of the following values:

- -1 (negative one). The object expressions reference the same object.
- 0 (zero). The object expressions do not reference the same object.

You can use this method to determine if an object variable is set to Nothing. For more information, see [Remove Object Method](#).

Format

```
objectExpression  
Is  
objectExpression
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>objectExpression</i>	Any valid object expression.

Example

For examples of using the Compare Object Expressions operator, see [Date and Time Methods](#) and [Get COM Object Method](#).

Is Expression a Date Method

The Is Expression a Date method determines if an expression evaluates to a date that Siebel VB allows. It returns one of the following values:

- -1 (negative one). The expression is a date variable type or a string that Siebel VB can interpret as a date.
- 0 (zero). The expression is not a date variable type or a string that Siebel VB can interpret as a date.

For more information, see [Variants](#).

Format

```
IsDate (expression)
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	Any valid expression.

Example

The following example adds a number to the current date value and determines if it is still a valid date. In this example, a valid date is in the range of January 1, 100, through December 31, 9999:

```
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

```

Is Object Of Class Method

The Is Object Of Class method determines if an object is of a given class. It returns one of the following values:

- -1 (negative one). The object is of the type that the className argument specifies.
- 0 (zero). The object is not of the type that the className argument specifies.

You can use this method only in an If statement.

Format

If Typeof *objectVariable* **Is** *className* **Then...**

You must use this format. You cannot combine this method with a Boolean operator.

The following table describes the arguments that you can use with this method.

Argument	Description
<i>objectVariable</i>	The object to be examined.
<i>className</i>	The class that Siebel CRM uses to compare the object.

You use the following format to determine if an object does *not* belong to a class:

```

If Typeof
objectVariable

Is
className

Then

[
  Perform some action
.]
Else
[
  Perform some action
.]
End If

```

Is Optional Argument Missing Method

The Is Optional Argument Missing method determines if an optional argument for a procedure is missing. It returns one of the following values:

- -1 (negative one). An optional argument is missing.
- 0 (zero). An optional argument is not missing.

Format

IsMissing (*argname*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>argname</i>	An optional argument for a subroutine, function, Siebel VB statement, or Siebel VB method.

Example

The following example prints a list of uppercase characters. The user determines the quantity printed. If the user must print every character, then this example calls the myfunc function without any argument. The function uses the Is Optional Argument Missing method to determine to print every uppercase character or to print only the quantity that the user specifies:

```
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
```

Is Variable Null Method

The Is Variable Null method determines if a variant variable contains a Null value. It returns one of the following values:

- -1 (negative one). The expression contains a Null value.
- 0 (zero). The expression does not contain a Null value.

A null variant does not contain data. It only identifies a result that is not valid or that is ambiguous. Null is not the same as Empty, which indicates that a variant is not yet set.

Format

IsNull(*expression*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	Any expression that contains a variable of data type variant.

Example

The following example asks for ten test score values and calculates the average. If any score is negative, then it sets the value to Null. This example uses the Is Variable Null method to reduce the total count of 10 scores to only the scores that contain a positive value before it calculates 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
```

Is Variable Numeric Method

The Is Variable Numeric method determines if the value of a variable is numeric. It returns one of the following values:

- 1 (negative one). The expression is a Numeric data type or is a string that Siebel VB can interpret as a number.
- 0 (zero). The expression is not a Numeric data type or is not a string that Siebel VB can interpret as a number.

If numeric input is required, then you can use the Is Variable Numeric method to determine if the value that the user provides is a valid number before converting the input to a numeric data type.

For more information, see [Variants](#).

Format

IsNumeric(*expression*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	Any valid expression.

Related Topics

[Get Variant Type Method](#)

Is Variable Set Method

The Is Variable Set method determines if a variable of data type variant is set. To indicate that it contains no data, every new variant defaults to an Empty type. This method returns one of the following values:

- -1 (negative one). The variant is set.
- 0 (zero). The variant is not set.

If you use an empty variant in a numeric expression or in a null string ("") in a string expression, then Siebel VB converts this empty variant to zero. For more information, see [Variants](#).

Format

IsEmpty(*expression*)

The following table describes the arguments that you can use with this method.

Argument	Description
<i>expression</i>	Any expression that identifies a variable of data type variant.

Example

The following example prompts the user for a series of test scores. It uses the Is Variable Set method to determine if Siebel CRM reached the maximum allowable limit. This method 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
```

Financial Methods

This topic describes financial methods. It includes the following topics:

- *Overview of Financial Methods*
- *Calculate Future Value Method*
- *Calculate Interest Method*
- *Calculate Interest Rate Method*
- *Calculate Internal Rate of Return Method*
- *Calculate Net Present Value Method*
- *Calculate Payment Method*
- *Calculate Principal Method*
- *Calculate Present Value Method*

Overview of Financial Methods

This topic includes an overview of financial methods.

Arguments You Can Use with Financial Methods

The following table describes arguments that you can use with financial methods. The topic for each method lists the arguments you can use with that method.

Argument	Description
<i>due</i>	An integer that specifies when payments are due. You can use one of the following values: • 0. End of each period. • 1. Beginning of each period.
<i>fv</i>	The future value of one of the following: • Final lump sum required in a savings plan • Final lump sum paid, which is 0 in a loan
<i>guess</i>	An estimate of the rate returned. This value <i>is typically</i> in the range of 0.1 (10 percent) through 0.15 (15 percent).
<i>nper</i>	The total number of payment periods.

Argument	Description
<i>per</i>	The payment period, in the range of 1 through the value that the <i>nper</i> argument contains.
<i>period</i>	The specific payment period, in the range 1 through the value that the <i>nper</i> argument contains.
<i>pmt</i>	The constant periodic payment for each period.
<i>pv</i>	The present value or the initial lump sum of one of the following: • Amount paid for an annuity • Amount received for a loan
<i>rate</i>	The interest rate for each period. For more information, see How Some Financial Methods Use the Rate Argument .
<i>valuearray</i>	An array that contains cash flow values. <i>This argument</i> must include at least one each of the following items: • A positive value that identifies a receipt • A negative value that identifies a payment You must represent payments and receipts in the exact sequence that the method must use to calculate them. The value that the method returns varies according to the modification that occurs in the sequence of cash flows.

How Some Financial Methods Use the Rate Argument

Some financial methods assume that the value that the rate argument includes is constant over the life of the annuity.

For example, if payments are on a monthly schedule, and if the annual percentage rate on the annuity or loan is 9%, then *the rate* is 0.0075 (.0075 equals .09 divided by 12).

Calculate Future Value Method

The Calculate Future Value method calculates, and then returns a number that identifies the future value of an investment, such as an annuity or a loan.

Format

```
FV(rate, nper, pmt, pv, due)
```

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates the future value of an annuity, according to terms that the user specifies:

```
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
```

Calculate Interest Method

The Calculate Interest method calculates, and then returns the interest portion of a payment for a given period of an annuity.

Format

```
IPmt(rate, period, nper, pv, fv, due)
```

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates 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
```

Calculate Interest Rate Method

The Calculate Interest Rate method calculates, and then returns the interest rate for each period for an annuity or a loan. This method is iterative. It improves a guess over multiple iterations until the result is within 0.00001 percent. If it does not converge to a result in 20 iterations, then it displays a failure message.

Format

`Rate(nper, pmt, pv, fv, due, guess)`

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates the interest rate on a 10 year, \$25,000 annuity that pays \$100 for each 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
```

Calculate Internal Rate of Return Method

The Calculate Internal Rate of Return method calculates, and then returns the internal rate of return for a stream of periodic cash flows. This method is iterative. It improves a guess over multiple iterations until the result is within 0.00001 percent. If it does not converge to a result in 20 iterations, then it displays a failure message.

Format

`IRR(valuearray( ), guess)`

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates an internal rate of return for a series of income and cost business transactions. It expresses this rate of return as an interest rate percentage. If the first value entered is a positive amount, then the IRR statement creates 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
```

Calculate Net Present Value Method

The Calculate Net Present Value method calculates, and then returns the net present value of an investment according to a stream of periodic cash flows and a constant interest rate. This method does the following:

- Returns the net present value in the *valuearray argument* according to the value that the *rate argument* contains.
- 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, then you must add this cash flow value to the result that this method returns. You must not include this value in the *valuearray argument*.

The value that the rate argument contains is the decimal equivalent of the discount rate. For example, if the discount rate is 12%, then the rate is 0.12.

Format

```
NPV(rate, valuearray( ))
```

The rate argument for this method describes the discount rate for each period. For a description of the valuearray argument, see [Arguments You Can Use with Financial Methods](#).

Calculate Payment Method

The Calculate Payment method calculates, and then returns a constant periodic payment amount for an annuity or a loan.

Format

```
Pmt(rate, nper, pv, fv, due)
```

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates the monthly payment for 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
```

Calculate Principal Method

The Calculate Principal method calculates, and then returns the principal portion of the payment for a given period of an annuity.

Format

`PPmt(rate, per, nper, pv, fv, due)`

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates 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
```

Calculate Present Value Method

The Calculate Present Value method calculates, and then returns the present value of a constant periodic stream of cash flows as in an annuity or a loan.

Format

`PV(rate, nper, pmt, fv, due)`

For more information, see [Arguments You Can Use with Financial Methods](#).

Example

The following example calculates the present value of a 10 year, \$25,000 annuity that pays \$1,000 for each 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
```

Conversion Methods

This topic describes conversion methods. It includes the following topics:

- [Convert Expression to Currency Method](#)
- [Convert Expression to Double-Precision Method](#)
- [Convert Expression to Integer Method](#)
- [Convert Expression to Long Method](#)
- [Convert Expression to Single-Precision Method](#)
- [Convert Expression to String Method](#)
- [Convert Expression to Variant Method](#)

Argument That You Can Use with Conversion Methods

The following table describes the arguments that you can use with conversion methods.

Argument	Description
<i>expression</i>	Any expression that evaluates to a number.

Convert Expression to Currency Method

The Convert Expression to Currency method converts the value that the *expression argument contains* to a currency. It returns *this currency*. Note the following:

- A number that does not fit in the currency data type causes an Overflow error.
- A string that cannot convert to a currency causes a Type Mismatch error.
- A variant that contains a null result causes an Illegal Use of Null error.

For more information, see [Overview of Data Types](#).

Format

`CCur(expression)`

For information about the arguments that you can use with this method, see [Argument That You Can Use with Conversion Methods](#).

Example

The following 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
```

Related Topics

- [Get ANSI String Method](#)
- [Convert Number to Date Method](#)

Convert Expression to Double-Precision Method

The Convert Expression to Double-Precision method converts the value that the *expression argument contains* to a double-precision number. It returns *this* number. Note the following:

- A string that cannot convert to a double-precision floating point number causes a Type Mismatch error.
- A variant that contains a null result causes an Illegal Use of Null error.

For more information, see [Numeric Data Types That Siebel VB Uses](#).

Format

`CDBl(expression)`

For information about the arguments that you can use with this method, see [Argument That You Can Use with Conversion Methods](#).

Example

The following 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
```

Convert Expression to Integer Method

The Convert Expression to Integer method converts the value that the *expression argument contains* to an integer and rounds the result. It returns this result. Note the following:

- After rounding, the result must reside in the range of negative 32767 to 32767. If it is not, then an error occurs.
- A string that cannot convert to an integer causes a Type Mismatch error.
- A variant that contains a null result causes an Illegal Use of Null error.

For more information, see [Numeric Data Types That Siebel VB Uses](#).

Format

`CInt(expression)`

For information about the arguments that you can use with this method, see [Argument That You Can Use with Conversion Methods](#).

Example

The following 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
```

Convert Expression to Long Method

The Convert Expression to Long method converts the value that the *expression argument contains* to a long number. It returns this number. Note the following:

- After rounding, the result must reside in the range of negative 32767 to 32767. If it is not, then an error occurs.
- A string that cannot convert to a long number causes a Type Mismatch error.
- A variant that contains a null result causes an Illegal Use of Null error.

For more information, see [Numeric Data Types That Siebel VB Uses](#).

Format

`CLng(expression)`

For information about the arguments that you can use with this method, see [Argument That You Can Use with Conversion Methods](#).

Example

The following example divides the national debt of the United States by the number of people who live in this country to find the amount of money each person must pay to retire this debt. It converts this number to a long integer, and then formats it as currency:

```
Sub Button_Click
    Dim debt As Single
    Dim msgtext
    Const Populace = 311000000
    debt = 14000000000000
    msgtext = "The $/citizen is: " & _
        Format(CLng(Debt/ Populace), "Currency")
End Sub
```

Convert Expression to Single-Precision Method

The Convert Expression to Single-Precision method converts the value that the *expression argument contains* to a single-precision, floating-point number. It returns this number. Note the following:

- After rounding, the result must reside in the range that the single-precision data type allows. If it is not, then an error occurs.
- A string that cannot convert to a single-precision number causes a Type Mismatch error.

For more information, see [Numeric Data Types That Siebel VB Uses](#).

Format

CSng(*expression*)

For information about the arguments that you can use with this method, see *Argument That You Can Use with Conversion Methods*.

Example

The following example calculates the factorial of a number. For more information, see *Exponential Method*:

```
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
```

Convert Expression to String Method

The Convert Expression to String method converts an expression to a string. It returns one of the values described in the following table depending on the data type of the expression.

<i>Data Type of Expression</i>	<i>Return Value</i>
Date	A string that contains a date.
Empty	A zero-length string ("").
Error	A string that contains the following information: Error error number
Null	A run-time error.
Other Numeric	A string that contains the number.

Format

CStr(*expression*)

For information about the arguments that you can use with this method, see [Argument That You Can Use with Conversion Methods](#).

Example

The following example uses the Convert Expression to String method to operate on a string that the user enters 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
```

Convert Expression to Variant Method

The Convert Expression to Variant method converts an expression to a variant. It returns this variant. It accepts any type of *expression*. It creates the same result that assigning the expression to a variant variable creates.

Format

CVar(*expression*)

For information about the arguments that you can use with this method, see [Argument That You Can Use with Conversion Methods](#).

Example

The following 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
```

COM Methods

This topic describes COM methods. It includes the following topics:

- [Assign COM Object Statement](#)
- [COM Object Class](#)

- *Create COM Object Method*
- *Get COM Object Method*
- *Initialize COM Object Method*

Assign COM Object Statement

The Assign COM Object statement assigns a COM object, such as an application, to a variable. In Siebel Tools, you can use it to create an instance of a Siebel object. It does not return a value. The Assign COM Object statement differs from the Let **statement**. The Let statement 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.

Format

```
Set  
  variableName  
=  
  objectExpression
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variableName</i>	An object variable or variant variable.
<i>objectExpression</i>	An expression that evaluates to an object. This object is typically one of the following items: • Function • Object member • Nothing

Including the Set Keyword When You Assign an Object Variable

If you do not include the Set keyword when you assign an object variable, then Siebel VB attempts to copy the default member of one object to the default member of another object. This situation typically results in the following run-time error:

```
' Incorrect code - tries to copy default member!  
COMObject = GetObject("", "spoly.cpoly")
```

Example 1

The following example uses the Assign COM Object statement:

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

COMObject.reset

Example 2

The following example creates an Opportunity business component outside the context of the user interface. The code prevents the user from deleting an account if there are opportunities associated with it. For more information about the Siebel VB methods and objects that this example uses, see *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
```

COM Object Class

The COM Object class provides access to a COM object. It does not return a value. To create a new object, you use the Dim statement to dimension a variable, and then set the variable to the return value of CreateObject or GetObject. For example:

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

You can use one of the following formats to reference a method or property of the new object:

```
objectvar.property
objectvar.method
```

For example:

COM.reset

Format

```
Dim variableName As Object
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>variableName</i>	The name of the object variable to declare.

Example

The following example uses the BusComp object class to declare the variables that Siebel VB uses to access the Account Contacts view in a 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
```

Create COM Object Method

The Create COM Object method creates a new COM object. It does not return a value.

Format

```
CreateObject(application.objectname)
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>application</i>	The name of the application.
<i>objectname</i>	The name of the object that this method uses.

Usage

To create an object, you use the Dim statement to declare an object variable, and then set the variable equal to the new object. For example:

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

You can use one of the following formats to reference a method or property of the object:

```
objectvar.property  
objectvar.method
```

For example:

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

You cannot display a modal or nonmodal form from a server application. A DLL that this method instantiates must be thread-safe.

To identify correct application and object names, see the documentation for your Web Client Automation Server application.

If a method passes the wrong number, order, or type of arguments to a COM object when it calls a COM object, then a 440 error message might occur.

Example 1

The following example uses the Create COM Object method to create a Microsoft Excel worksheet. It then edits and saves this worksheet:

```
Sub BtnExcel_Click  
  Dim oWorkSheet As Object  
  Dim sfileName As String  
  Set oWorkSheet = CreateObject("Excel.Sheet")  
  If oWorkSheet Is Nothing then  
    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 (sfileName)
' Close Excel with the Quit method on the Application object
oWorksheet.Application.Quit
' Clear the object from memory
Set oWorksheet = Nothing
End Sub
```

Example 2

The following example uses the Create COM Object method to create a Microsoft Word document. It then edits and saves this 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
```

Get COM Object Method

The Get COM Object method returns the COM object that the *pathname argument* or the *class* argument identifies. To assign a variable to an object for use in a Visual Basic procedure, you dimension a variable as an object, and then use the Get COM Object method with the Assign COM Object statement.

The examples in this topic reference the SiebelAppServer object, which you define as an object type in your external Visual Basic environment.

Format A

```
GetObject(pathname)
```

Format B

```
GetObject(pathname, class)
```

Format C

```
GetObject(, class)
```

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>pathname</i>	The full path and file name for the object to get.
<i>class</i>	A string that contains the class of the object.

Usage for Format A

You can use format A to access a COM object that is stored in a file. For example, the following code dimensions a variable as an object and assigns the payables.xls object to this variable. The Payables.xls file is located in the My Documents directory:

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

If the Siebel application supports accessing component objects in the file, then you can append an exclamation point and a component object name to the file name. For example:

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

Usage for Format B

You can use format B to access a COM object of a particular class that is stored in a file. The class argument uses the following format:

```
appName.objectType
```

where:

- **appName** is the name of the application that provides the object
- **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")
```

Usage for Format C

You can use format C to access the active COM object of a particular class. For example:

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

If you use format C with a null string ("") in the pathname argument, then Siebel VB returns a new object instance of the class that you specify in the class argument. The preceding example gets an open instance of the Siebel application. The following example instantiates the Siebel application in memory, independent of the user interface:

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

Example

The following example opens an Excel worksheet and places the contents of the Name field of the active business component in this worksheet. 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
```

Initialize COM Object Method

The Initialize COM Object method allocates and initializes a new COM object. It does not return a value.

In the Declare Variable statement, the New argument instructs Siebel VB to allocate and set a new COM object the first time it encounters the object that the *objectVar argument* identifies. If the code does not reference this object, then it does not allocate a new object.

The Initialize COM Object method does not create a COM object until the first time Siebel eScript uses this object. If Siebel eScript never uses this object, then this method does not create the object. The Create COM Object method creates the object as soon as you call this method. For more information, see [Create COM Object Method](#).

If the `objectVar` argument contains `Nothing`, and if you declare an object variable, and if you reference this object again, then `New` allocates a second object. For more information, see [Remove Object Method](#).

Format

```
Set  
  objectVar  
    = New  
      className  
Dim  
  objectVar  
    As New  
      className
```

The following table describes the arguments that you can use with this method.

Argument	Description
<code>objectVar</code>	The COM object to allocate and initialize.
<code>className</code>	The class to assign to the object.

Error Handling Methods

This topic describes error handling methods. It includes the following topics:

- [Get Error Code Method](#)
- [Get Error Code Line Method](#)
- [Get Error Message Method](#)
- [On Error Method](#)
- [Resume Statement](#)
- [Set Error Code Method](#)
- [Simulate Error Method](#)

Get Error Code Method

The `Get Error Code` method returns the error code of the last Visual Basic error handled. You can use the `Err` statement or the `Error` statement to set the value for this method.

You cannot use the `Get Error Code` method to view a Siebel VB error. Instead, you use the appropriate method for the COM Siebel interface that you are using. For more information, see [Siebel Object Interfaces Reference](#).

For more information, see [Error Code and Error Text for Siebel VB Errors](#).

Format

Err

This method does not include arguments.

Example

For examples, see [Get Error Code Line Method](#) and [Get Error Message Method](#).

Get Error Code Line Method

The Get Error Code Line method returns a number that identifies the code line where an error occurred. If you use a Resume statement or an On Error statement after you use this method, then this method sets the return value to 0. To maintain the value of the line number, you must assign it to a variable. You can use the Error statement to set this return value.

Format

Erl

This method does not include arguments.

Example

The following example uses the Err statement to print the error number and the Erl statement to print the line number if an error occurs during an attempt to open a file. Siebel VB assigns line numbers, starting with 1. In this example the **Sub Button_Click** statement is line 1:

```
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
```

Get Error Message Method

The Get Error Message method returns the error message that corresponds to an error code. If it does not find an error message that matches the error code, then it returns a null string (""). For more information, see [Error Code and Error Text for Siebel VB Errors](#).

Format

Error[\$] [(errornumber)]

For information about the dollar sign, see *Usage of the Dollar Sign*.

The following table describes the arguments that you can use with this method.

Argument	Description
<i>errornumber</i>	An integer in the range of 1 through 32,767 that identifies an error code. If you do not include this argument, then Siebel VB returns the error message of the run-time error that occurred most recently.

Example

The following example uses the Err statement to print the error number and the text of the error. If an error occurs during an attempt to open a file, then it uses the Error\$ 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 " & Err & ": " & Error$
  Resume done
End Sub
```

On Error Method

The On Error method identifies the location of code that handles an error. It does not return a value. You can also use it to disable code that handles the error. If you do not use the On Error method, and if a run-time error occurs, then Siebel VB stops running code.

Format

```
On Error {GoTo label | Resume Next | GoTo 0}
```

The following table describes that parts of an On Error statement. You can include only one of these parts.

Part	Description
GoTo <i>label</i>	Identifies the code that handles the error that starts at the <i>label argument</i> . If this label does not reside in the same procedure as the On Error statement, then Siebel VB creates an error message.
Resume Next	Identifies the code to run after handling the error. It typically identifies the code that occurs immediately after the statement that caused the error. You can use the Err statement to get the error code of the run-time error at this point in the error handling.
GoTo 0	Disables any error handler that is enabled.

Usage

Note the following:

- If an `On Error GoTo label` statement references an error handler, then that error handler is enabled.
- If an error handler is enabled, and if a run-time error occurs, then Siebel VB gives control to the code that includes this error handler. The error handler remains active from the time the run-time error is handled until code flow encounters a `Resume` statement in the error handler.
- If another error occurs while the error handler is active, then Siebel VB searches for the error handler in the procedure that called the current procedure:
- If it find this error handler, then it stops the current procedure and activates the error handler in the calling procedure.
- If it does not find this error handler, then it searches for a handler that resides in the procedure that called the calling procedure, and so on.

Because Siebel VB searches in the caller for an error handler, it ignores any `On Error` statements that exist in the original error handler.

- Running an `End Sub` or `End Method` statement while an error handler is active creates a `No Resume` error. You can use the `Exit Sub` or `Exit Method` statement to end the error condition and exit the current procedure.

Example

The following example prompts the user for a drive and directory name. It uses the `On Error` method to handle an entry that is not valid:

```
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
```

Resume Statement

The Resume statement stops the code that handles an error, and then passes control to the statement that immediately follows the statement where the error occurred. It does not return a value.

If you use Resume [0], then control passes to the statement where the error occurred.

The location of the error handler that handles the error determines where code resumes:

- If the error is located in the same procedure as the error handler, then code flows to the statement that caused the error.
- If the error is not located in the same procedure as the error handler, then code flows to the statement that last called the procedure that contains the error handler.

Format A

Resume **Next**

Format B

Resume *label*

Format C

Resume [0]

Arguments

The following table describes the arguments that you can use with this method.

Argument	Description
<i>label</i>	The label that identifies the code line to go to after handling an error.

Set Error Code Method

The Set Error Code method sets a run-time error code. It does not return a value. You can use it to send error information between procedures.

Format

Err = *errornumber*

The following table describes the arguments that you can use with this method.

Argument	Description
<i>errornumber</i>	An integer in the range of 1 through 32,767 that identifies an error code, or a 0 if no error occurs.

Example

The following example creates an error code of 10000 and displays an error message if the user does not enter a customer name in reply to a prompt. It uses the Err statement to clear any previous error codes before it runs the loop for the first time. It also clears the error to allow the user to try again:

```
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
```

For another example, see the following.

Simulate Error Method

The Simulate Error method simulates the occurrence of an error. Note the following:

- If an Error statement runs, and if code to handle the error does not exist, then Siebel VB creates an error message and stops code from running.
- If an Error statement specifies an error code that Siebel VB does not use, then it displays the following error message:

User-defined error

Format

```
Error errornumber
```

The following table describes the arguments that you can use with this method.

Argument	Description
<i>errornumber</i>	An integer in the range of 1 through 32,767 that identifies an error code. If Siebel VB already uses an error number that the <i>errornumber argument identifies</i> , then the Error statement simulates an occurrence of that error.

Using a Custom Error Code

A custom error code must use a value that is greater than any value that Siebel VB uses for a predefined error code. To avoid using a predefined error code, it is recommended that the value you use for a custom error code start with 32,767. For subsequent custom error codes, you can work down from 32,767.

Do not use any error code in the range of 4000 through 4999. These are predefined codes for Siebel VB methods. For more information, see *Siebel Object Interfaces Reference*.

5 Quick Reference for Siebel VB Methods

Quick Reference for Siebel VB Methods

This quick reference section lists Siebel VB methods. It includes the following topics:

- *Disk and Directory Control Quick Reference*
- *File Control Quick Reference*
- *File Input and Output Quick Reference*
- *Code Setup and Control Quick Reference*
- *Code Control Statements Quick Reference*
- *Variable Manipulation Quick Reference*
- *Strings Quick Reference*
- *Arrays Quick Reference*
- *Math Operations Quick Reference*
- *Date and Time Quick Reference*
- *ODBC Quick Reference*
- *Object Querying Quick Reference*
- *Financials Quick Reference*
- *Conversions Quick Reference*
- *COM Object Quick Reference*
- *Error Handling Quick Reference*

Disk and Directory Control Quick Reference

The following table lists methods that you can use to control disks and directories.

Statement	Purpose	Reference
ChDir	Modifies the default directory for a drive.	<i>Change Directory Method</i>
ChDrive	Modifies the default drive.	<i>Change Drive Method</i>
CurDir	Returns the current directory for a drive.	<i>Get Current Directory Method</i>
MkDir	Creates a directory.	<i>Create Directory Method</i>

Statement	Purpose	Reference
RmDir	Removes a directory.	<i>Remove Directory Method</i>

File Control Quick Reference

The following table lists methods that you can use to control files.

Statement	Purpose	Reference
Close	Closes a file.	<i>Close File Method</i>
Dir	Returns a file name that matches a pattern.	<i>Get File Names Method</i>
FileAttr	Returns the file mode or the operating system handle for an open file.	<i>Get File Mode Method</i>
FileCopy	Copies a file.	<i>Copy File Method</i>
FileDateTime	Returns the last modification date and time of a file.	<i>Get File Date Method</i>
FileLen	Returns the length of a file.	<i>Get File Length 2 Method</i>
FreeFile	Returns the next unused file number.	<i>Get Free File Number Method</i>
GetAttr	Returns attributes of a file, directory, or volume label.	<i>Get File Attributes Method</i>
Kill	Deletes a file.	<i>Delete File Method</i>
Lock	Controls access to an open file.	<i>Lock File Method</i>
Lof	Returns the length of a file in bytes.	<i>Get File Length Method</i>
Name	Renames a file or copies a file from one directory to another directory.	<i>Rename File Method</i>
Open	Opens a file.	<i>Open File Method</i>
Reset	Closes open files and writes to disk any data that currently resides in buffers.	<i>Close All Files Method</i>
SetAttr	Sets attribute information for a file.	<i>Set File Attributes Method</i>

Statement	Purpose	Reference
Unlock	Controls access to some or all of an open file by other processes.	<i>Unlock File Method</i>

File Input and Output Quick Reference

The following table lists methods that you can use to manipulate data in a file.

Statement	Purpose	Reference
Eof	Determines if the end of an open file has been reached.	<i>End of File Method</i>
Get	Reads the content of a file, and then places this content in a variable.	<i>Force Explicit Declaration Statement</i>
Input	Returns a string of characters from a file.	<i>Get Characters From File Method</i>
Input filenumber, variable	Reads data from a file, and then saves this data to different variables.	<i>Parse File Contents Method</i>
Line Input	Reads a line from a sequential file, and then saves it in a string variable.	<i>Get Line From File Method</i>
Loc	Returns the current offset of a file.	<i>Get File Offset Method</i>
Print	Prints data to a file or to the screen.	<i>Print Data to File Method</i>
Put	Writes a variable to a file.	<i>Write Variable to File Method</i>
Seek filenumber	Returns the current position for a file.	<i>Get File Position Method</i>
Seek filenumber, position	Sets the position of the next read or write operation in an open file.	<i>Set File Position Method</i>
Spc	Prints a number of spaces.	<i>Print Spaces Method</i>
Tab	Sets the current print position.	<i>Set Print Position Method</i>
Width	Sets the output line width for an open file.	<i>Set File Width Method</i>

Statement	Purpose	Reference
Write	Writes data to a sequential file.	<i>Write Data to File Method</i>

Code Setup and Control Quick Reference

The following table lists methods that you can use to perform setup tasks and to control the flow of logic in Siebel VB code.

Statement	Purpose	Reference
Shell	Starts a Microsoft Windows application.	<i>Call Application Method</i>
Call	Transfers control to a subroutine.	<i>Call Subroutine Method</i>
Sub name	Creates a subroutine.	<i>Create Subroutine Method</i>
Function	Creates a function.	<i>Create Function Method</i>
Type userType	Declares a custom data type.	<i>Declare Custom Data Type Method</i>
Declare Sub Declare Function	Declares a procedure or a function.	<i>Declare Procedure Method</i>
Const constantName	Declares a symbolic constant.	<i>Declare Symbolic Constant Method</i>
Environ	Returns a string from the operating system environment	<i>Get Environment Setting Method</i>
Nothing	Sets an object variable to not reference an object.	<i>Remove Object Method</i>
AppActivate	Sends keystrokes to a Microsoft Windows application.	<i>Send Keystrokes Method</i>
Clipboard	Accesses the Microsoft Windows clipboard.	<i>Use Clipboard Methods</i>

Code Control Statements Quick Reference

The following table lists statements you can use to control the flow of logic in Siebel VB code.

Statement	Purpose	Reference
_	Treats the next line as a continuation of the current line.	Not applicable.
Command	Returns the command line specified when the MAIN sub runs.	Not applicable.
Do Loop	Controls repetitive actions.	<i>Do Loop Statement</i>
Exit	Causes the current procedure or loop structure to return.	<i>Exit Statement</i>
For Next	Loops a fixed number of times.	<i>For Next Statement</i>
Go To	Sends control to a statement.	<i>Go To Statement</i>
Go To Label	Branches to one of multiple labels depending on value.	<i>Go To Label Statement</i>
If...Then... Else	Branches on a conditional value.	<i>If Then Else Statement</i>
Me	Gets the current object.	<i>Me Statement</i>
Rem or '	Treats the remainder of the line as a comment	<i>Rem Statement</i>
Select Case	Runs one of a series of statement blocks.	<i>Select Case Statement</i>
Stop	Stops code from running.	<i>Stop Statement</i>
While Wend	Controls repetitive actions.	<i>While Wend Statement</i>

Variable Manipulation Quick Reference

The following table lists methods to manipulate variables.

Statement	Purpose	Reference
DefType	Sets the default data type for one or more variables.	Set Variable Data Type Statement
Dim	Declares a variable.	Declare Variable Statement
Global	Declares a global variable.	Declare Global Variable Statement
Let	Assigns a value to a variable.	Assign Expression to Variable Statement
Null	Sets a variant variable to a value of Null.	Set Variant Variable to Null Method
Option Explicit	Forces you to explicitly declare every variable in a module.	Force Explicit Declaration Statement
Static	Declares a variable and allocates storage space for this variable.	Declare Static Variable Statement
VarType	Returns the type of data stored in a variant.	Get Variant Type Method
With	Runs a series of statements on a variable.	Modify Variable Statement

Strings Quick Reference

The following table lists methods to manipulate strings.

Statement	Purpose	Reference
Chr	Converts a character code to a string.	Get ANSI String Method
Format	Returns an <i>expression</i> in a <i>format</i> that you specify.	Set String Format Method
InStr	Returns the position of one string in another string.	Get Substring Position Method
LCase	Converts a string to lower case.	Convert String to Lowercase Method
Left	Returns a string copied from the beginning of another string.	Get Left String Method

Statement	Purpose	Reference
Len	Returns the length of a string or size of a variable.	<i>Get String Length Method</i>
Like	Compares the contents of two strings.	<i>Compare Strings Operator</i>
Lset	Copies one string to another string or assigns a custom variable to another variable.	<i>Copy String Method</i>
LTrim	Removes leading spaces from a string.	<i>Remove Spaces From String Method</i>
Mid	Returns a portion of a string.	<i>Get Substring Method</i>
Mid = string	Replaces part or all of one string with another string.	<i>Replace String Method</i>
Option Compare	Specifies the default method for string comparisons to case-sensitive or not case-sensitive.	<i>Set String Comparison Method</i>
Right	Returns the right portion of a string.	<i>Get Right String Method</i>
Rset	Right-justifies one string in another string.	<i>Right-Justify String Method</i>
RTrim	Removes trailing spaces from a string.	<i>Trim Trailing Spaces From String Method</i>
Space	Returns a string of spaces.	<i>Get a String of Spaces Method</i>
Str	Returns the string representation of a number.	<i>Convert Number to String Method</i>
StrComp	Compares two strings.	<i>Compare Strings Method</i>
String	Returns a string that consists of a repeated character.	<i>Get Repeated Character String Method</i>
Trim	Removes leading and trailing spaces from a string.	<i>Trim Spaces From String Method</i>
UCase	Converts a string to upper case.	<i>Convert String to Uppercase Method</i>
Val	Returns the numeric value of the first number that it finds in a string.	<i>Get First Number From String Method</i>

Arrays Quick Reference

The following table lists methods for manipulating arrays.

Statement	Purpose	Reference
Erase	Erases the contents of an array.	<i>Erase Array Method</i>
LBound	Gets the lower boundary of an array.	<i>Get Array Lower Boundary Method</i>
Option Base	Specifies the default lower boundary to use for an array.	<i>Set Array Lower Boundary Method</i>
ReDim	Declares an array and reallocates memory.	<i>Declare Array Method</i>
UBound	Gets the upper boundary of an array.	<i>Get Array Upper Boundary Method</i>

Math Operations Quick Reference

The following table lists methods to perform mathematical operations.

Statement	Purpose	Reference
Abs	Returns the absolute value of a number.	<i>Get Absolute Value Method</i>
Asc	Returns an integer that corresponds to an ANSI character code.	<i>Get ANSI Integer Method</i>
Atn	Returns the arctangent of a number.	<i>Get Arctangent Method</i>
Cos	Returns the cosine of an angle.	<i>Get Cosine Method</i>
Exp	Returns the value of e raised to a power.	<i>Convert Number to Date Method</i>
Fix	Removes the fractional part of a number.	<i>Get Rounded Integer Method</i>
Hex	Returns the hexadecimal representation of a number.	<i>Get Hexadecimal Method</i>

Statement	Purpose	Reference
Int	Returns the integer part of a number.	<i>Get Integer Method</i>
Log	Returns the natural logarithm of a value.	<i>Get Logarithm Method</i>
Oct	Returns the octal representation of a number.	<i>Get Octal Method</i>
Randomize	Creates a starting value for the random number generator.	<i>Randomize Method</i>
Rnd	Returns a random number.	<i>Get Random Number Method</i>
Sgn	Returns a value that identifies the sign of a number.	<i>Get Number Sign Method</i>
Sin	Returns the sine of an angle.	<i>Get Sine Method</i>
Sqr	Returns the square root of a number.	<i>Get Square Root Method</i>
Tan	Returns the tangent of an angle.	<i>Get Tangent Method</i>

Date and Time Quick Reference

The following table lists methods for date and time information.

Statement	Purpose	Reference
Date	Converts an expression to the data type variant of type date.	<i>Convert Number to Date Method</i>
Date = expression	Sets the computer date.	<i>Set Date Method</i>
DateSerial	Converts a number to a date.	<i>Convert Serial Number to Date Method</i>
DateValue	Converts a string to a date.	<i>Convert String to Date Method</i>
Day	Returns the day component of a date and time value.	<i>Extract Day From Date-Time Value Method</i>
Hour	Returns the hour of day component of a date and time value.	<i>Extract Hour From Date-Time Value Method</i>

Statement	Purpose	Reference
Minute	Returns the minute component of a date and time value.	<i>Extract Minute From Date-Time Value Method</i>
Month	Returns the month component of a date and time value.	<i>Extract Month From Date-Time Value Method</i>
Now	Returns the current date and time.	<i>Get Current Date and Time Method</i>
Second	Returns the second component of a date and time value.	<i>Extract Second From Date-Time Value Method</i>
Time	Returns the current time.	<i>Get Current Time Method</i>
Time = expression	Sets the current time.	<i>Set Time Method</i>
Timer	Returns the number of seconds since midnight.	<i>Get Current Seconds Method</i>
TimeSerial	Returns the time for a specific hour, minute, and second.	<i>Get Serial Time Method</i>
TimeValue	Converts a string to time.	<i>Convert String to Time Method</i>
Weekday	Returns the day of the week for the specified date and time value.	<i>Extract Weekday From Date-Time Value Method</i>
Year	Returns the year component of a date and time value.	<i>Extract Year From Date-Time Value Method</i>

ODBC Quick Reference

The following table lists methods that you can use with ODBC.

Statement	Purpose	Reference
SQLClose	Disconnects from an ODBC data source connection.	<i>ODBC Close Connection Method</i>
SQLError	Returns a detailed error message for an error that occurs during an ODBC method call.	<i>ODBC Get Errors Method</i>
SQLExecQuery	Runs an SQL statement.	<i>ODBC Run Query Method</i>

Statement	Purpose	Reference
SQLGetSchema	Gets information about data sources, databases, users, owners, and so on.	<i>ODBC Get Schema Method</i>
SQLOpen	Establishes a connection to an ODBC data source.	<i>ODBC Open Connection Method</i>
SQLRequest	Makes a connection to a data source, runs an SQL statement, returns the results.	<i>ODBC Run Query and Get Results Method</i>
SQLRetrieve	Returns the results of a select statement that SQLExecQuery runs.	<i>ODBC Get Query Results Method</i>
SQLRetrieveToFile	Gets the results of an SQL query and stores them in a file.	<i>ODBC Save Results to File Method</i>

Object Querying Quick Reference

The following table lists methods that you can use to query an object.

Statement	Purpose	Reference
If Typeof	Determines if an object is of a given class.	<i>Is Object Of Class Method</i>
IsDate	Determines if an expression evaluates to a date that Oracle's Siebel VB allows.	<i>Is Expression a Date Method</i>
IsEmpty	Determines if a variant has been set.	<i>Is Variable Set Method</i>
IsMissing	Determines if an optional argument for a procedure is missing.	<i>Is Optional Argument Missing Method</i>
IsNull	Determines if a variant contains a NULL value.	<i>Is Variable Null Method</i>
IsNumeric	Determines if a value is a valid number.	<i>Is Variable Numeric Method</i>

Financials Quick Reference

The following table lists methods that you can use to calculate financial information.

Statement	Purpose	Reference
FV	Returns future value of a cash flow stream.	<i>Calculate Future Value Method</i>
IPmt	Returns interest payment for a given period.	<i>Calculate Interest Method</i>
IRR	Returns internal rate of return for a cash flow stream.	<i>Calculate Internal Rate of Return Method</i>
NPV	Returns net present value of a cash flow stream.	<i>Calculate Net Present Value Method</i>
Pmt	Returns a constant payment for each period for an annuity.	<i>Calculate Payment Method</i>
PPmt	Returns principal payment for a given period.	<i>Calculate Principal Method</i>
PV	Returns present value of a future stream of cash flows.	<i>Calculate Present Value Method</i>
Rate	Returns interest rate for each period.	<i>Calculate Interest Rate Method</i>

Conversions Quick Reference

The following table lists methods that you can use to convert a value.

Statement	Purpose	Reference
CCur	Converts a value to currency.	<i>Convert Expression to Currency Method</i>
CDbl	Converts a value to a double-precision floating point number.	<i>Convert Expression to Double-Precision Method</i>
CLnt	Converts a value to an integer.	<i>Convert Expression to Integer Method</i>
CLng	Converts a value to a long number.	<i>Convert Expression to Long Method</i>
CSng	Converts a value to single-precision, floating point number.	<i>Convert Expression to Single-Precision Method</i>
CStr	Converts a value to a string.	<i>Convert Expression to String Method</i>
CVar	Converts a number or string to a variant.	<i>Convert Expression to Variant Method</i>
CVDate	Converts a value to a date.	<i>Convert Number to Date Method</i>

Statement	Purpose	Reference

COM Object Quick Reference

The following table lists methods for COM objects.

Statement	Purpose	Reference
CreateObject	Creates a new COM object.	<i>Create COM Object Method</i>
Dim As Object	A class that provides access to a COM object.	<i>COM Object Class</i>
GetObject	Gets a COM object from a file or gets the active COM object for a COM class.	<i>Get COM Object Method</i>
New	Allocates and initializes a new COM object.	<i>Initialize COM Object Method</i>
Set	Assigns a COM object to a variable.	<i>Assign COM Object Statement</i>

Error Handling Quick Reference

The following table lists methods for error handling.

Statement	Purpose	Reference
Erl	Returns a number that identifies the code line where an error occurred.	<i>Get Error Code Line Method</i>
Err	Returns the error code of the last Visual Basic error handled.	<i>Get Error Code Method</i>
Err = errornumber	Sets a run-time error code.	<i>Set Error Code Method</i>
Error	Returns the error message that corresponds to an error code.	<i>Get Error Message Method</i>
Error errornumber	Simulates the occurrence of an error.	<i>Simulate Error Method</i>

Statement	Purpose	Reference
On Error	Identifies the location of code that handles an error.	<i>On Error Method</i>
Resume	Stops the code that handles an error.	<i>Resume Statement</i>