

Appendix E provides a quick reference for the functions and methods that are discussed in this book. Each entry provides the general format of the function or method and lists the types of exceptions that may be caused. (This is not a complete list of all the Visual Basic functions and methods. For information on a function or method that is not listed here, consult the *Visual Basic Help* system.)

#### Add method—collection

```
Object.Add(Item [, Key] [, Before] [,After])
```

`Object` is the name of an object variable that references a collection. The `Item` argument is the object, variable, or value that is to be added to the collection. `Key` is an optional string expression that is associated with the item and can be used to search for it. (`Key` must be unique for each member of a collection.) `Before` is an optional argument that can be used when you want to insert an item before an existing member. It can be either a string that identifies the existing member's key or a number specifying the existing member's index. The optional `After` argument works just like the `Before` argument, except that the new item is inserted after the existing member. If you do not specify a `Before` or `After` value, the new member will be added to the end of the collection. You cannot use both `Before` and `After`.

Exception	Condition
<code>ArgumentException</code>	Occurs when the key value already exists in the collection, or when both the <code>Before</code> and <code>After</code> arguments are provided.

**Add method—DataSet**

```
DataSet.Table.Rows.Add(ObjectVar)
```

*DataSet* is the name of the dataset and *ObjectVar* is the name of the object variable that references the new row. After the statement executes, the row referenced by *ObjectVar* will be added to the table.

Exception	Condition
ArgumentNullException	The row object referenced by <i>ObjectVar</i> is <i>Nothing</i> .
ArgumentException	The row is already inserted into the table, or belongs to another table.
ConstraintException	This action violates a constraint.
NotNullAllowedException	A column in the row is null and the column does not allow null values.

**Add method—ListBox control**

```
ListBox.Items.Add(Item)
```

*ListBox* is the name of the *ListBox* control. *Item* is the item that is to be added to the *Items* property.

Exception	Condition
SystemException	The system does not have enough available memory to accommodate the addition of this item.

**AppendText—System.IO.File**

```
System.IO.File.AppendText(Filename)
```

*Filename* is a string or a string variable specifying the name of the file on the disk, including optional path information. This method creates the file specified by *Filename* and returns the address of a *StreamWriter* object that may be used to write data to the file. If the file already exists when this method executes, data written to the file will be appended to the end of the existing data. If the file does not already exist, it will be created.

Exception	Condition
SecurityException	The calling application does not have the required permission.
ArgumentException	The filename or path is invalid.
ArgumentNullException	<i>Filename</i> is set to <i>Nothing</i> .
PathTooLongException	The path in <i>Filename</i> exceeds the maximum limit for the system.
DirectoryNotFoundException	The directory specified does not exist.
FileNotFoundException	The file specified does not exist.
NotSupportedException	<i>Filename</i> contains a colon (:) in the middle of the string.

**Chr function**

`Chr(CharacterCode)`

*CharacterCode* is an integer character code. The function returns the character that corresponds to the character code.

Exception	Condition
<code>ArgumentException</code>	The character code is less than <code>-32768</code> or <code>&gt; 65535</code> .

**Clear method—ListBox control**

`ListBox.Items.Clear()`

Erases all the items in a list box's `Items` property.

**Clear method—TextBox control**

`TextBox.Clear()`

Clears the `Text` property of a `TextBox` control.

**Clipboard.GetText—Clipboard**

`My.Computer.Clipboard.GetText(TextDataFormat.Text)`

This form of the method returns the text currently stored in the clipboard. If there is no text in the clipboard, it returns an empty string.

**Close method—BinaryReader class**

`ObjectVar.Close()`

*ObjectVar* is the name of a `BinaryReader` object variable. After the method executes, the file opened with *ObjectVar* is closed.

**Close method—BinaryWriter class**

`ObjectVar.Close()`

*ObjectVar* is the name of a `BinaryWriter` object variable. After the method executes, the file opened with *ObjectVar* is closed.

**Close method—Form class**

`Me.Close()`

Closes a form. The `Me` keyword references the current instance of the form. A form executing the `Me.Close()` statement is calling its own `Close` method.

Exception	Condition
<code>InvalidOperationException</code>	The form was closed while a handle was being created.

**Close method—StreamReader class**

`ObjectVar.Close()`

*ObjectVar* is the name of a `StreamReader` object variable. After the method executes, the file opened with *ObjectVar* is closed.

**Close method—StreamWriter class**

```
ObjectVar.Close()
```

*ObjectVar* is the name of a StreamWriter object variable. After the method executes, the file opened with *ObjectVar* is closed.

**CreateText method—System.IO.File**

```
System.IO.File.CreateText(Filename)
```

*Filename* is a string or a string variable specifying the name of the file. This method creates the file specified by *Filename* and returns the address of a StreamWriter object that may be used to write data to the file. If the file already exists when this method executes, its contents are erased.

Exception	Condition
SecurityException	The calling application does not have the required permission.
ArgumentException	The filename or path is invalid.
ArgumentNullException	<i>Filename</i> is set to Nothing.
PathTooLongException	The path in <i>Filename</i> exceeds the maximum limit for the system.
DirectoryNotFoundException	The directory specified does not exist.
FileNotFoundException	The file specified does not exist.
NotSupportedException	<i>Filename</i> contains a colon (:) in the middle of the string.

**DateAdd function**

```
DateAdd(Interval, Number, DateValue)
```

Adds a time interval to a date. *Interval* is a value that represents the interval we are adding to the date. For example, the value DateInterval.Second specifies that we are adding seconds to the date, and the value DateInterval.Hour specifies that we are adding hours to the date. *Number* is the number of intervals we are adding. *DateValue* is the Date value that we are adding to. The function returns the resulting Date value.

Exception	Condition
ArgumentException	The <i>Interval</i> argument is invalid.
ArgumentOutOfRangeException	The resulting date is before 00:00:00 on January 1 of year 1, or later than 23:59:59 on December 31, 9999.
InvalidCastException	The <i>DateValue</i> argument cannot be converted to a Date data type.

**EOF Function**

```
EOF(FileNumber)
```

Tests a random-access file to determine if the end of the file has been reached. *FileNumber* is the number of the file to test. The function returns True if the end of the file has already been reached, or False otherwise.

Exception	Condition
IOException	<i>FileNumber</i> does not exist, or the file's mode is invalid.

### FileClose method

```
FileClose(FileNumber)
```

Closes a random-access file. *FileNumber* is the file number associated with the file.

### FileOpen method

```
FileOpen(FileNumber, Filename, OpenMode.Random, _  
        AccessMode, OpenShare.Default, RecordLength)
```

Opens a random-access file. *FileNumber* is an integer that you use to identify the file. *Filename* is the name of the file, which can include the file's path. *AccessMode* specifies how the file will be accessed, and can be one of the following values: `OpenAccess.Default`, `OpenAccess.Read`, `OpenAccess.ReadWrite`, or `OpenAccess.Write`. Note that a random-access file can be opened for both reading and writing with `OpenAccess.ReadWrite`. *RecordLength* is the size of the records that will be stored in the file.

Exception	Condition
ArgumentException	An invalid value for <i>AccessMode</i> was specified, or a write-only file was opened for input, or a read-only file was opened for output, or <i>RecordLength</i> is negative (and not -1).
IOException	<i>FileNumber</i> is < -1 or > 255 or already in use. This exception is also caused when <i>Filename</i> is invalid or the file is already opened.

### FileGet method

```
FileGet(FileNumber, Item, RecordNumber)
```

Reads a record from a random-access file. *FileNumber* is the number of the file. This is the same number you specified as the *FileNumber* argument with the *FileOpen* method. *Item* is a variable that will hold the record. *RecordNumber* is the number of the record to read.

Exception	Condition
ArgumentException	<i>RecordNumber</i> is < 1 (and not -1).
IOException	<i>FileNumber</i> does not exist or file mode is invalid for this operation.

### FilePut method

```
FilePut(FileNumber, Item, RecordNumber)
```

Writes a record to a random-access file. *FileNumber* is the number of the file. This is the same number you specified as the *FileNumber* argument with the *FileOpen* method. *Item* is the record to be written. *RecordNumber* is the number of the record to write.

Exception	Condition
<code>ArgumentException</code>	<i>RecordNumber</i> is < 1 (and not -1).
<code>IOException</code>	<i>FileNumber</i> does not exist or file mode is invalid for this operation.

### Fill method—`DataAdapter`

```
DataAdapter.Fill(DataSet)
```

*DataAdapter* is the name of a data adapter object, and *DataSet* is the name of the dataset object to fill. After this method executes, the dataset object will contain a table of data.

### DrawString method

```
e.Graphics.DrawString(String, New Font(FontName, _
    Size, Style), Brushes.Black, HPos, VPos)
```

*String* is the string that is to be printed. *FontName* is a string holding the name of the font to use. *Size* is the size of the font, measured in points. *Style* is the font style. Typically, the valid values are `FontStyle.Bold`, `FontStyle.Italic`, `FontStyle.Regular`, `FontStyle.Strikeout`, and `FontStyle.Underline`. *HPos* is the horizontal position of the output. This is the distance of the output, in points, from the left margin of the paper. *VPos* is the vertical position of the output. This is the distance of the output, in points, from the top margin of the paper. The `Brushes.Black` argument specifies that output should be printed in black.

### Exists method—`System.IO.File`

```
System.IO.File.Exists(Filename)
```

*Filename* is the name of a file, which may include the path. The method returns `True` if the file exists, or `False` if the file does not exist.

Exception	Condition
<code>ArgumentException</code>	The filename or path is invalid.
<code>ArgumentNullException</code>	<i>Filename</i> is set to <code>Nothing</code> .
<code>PathTooLongException</code>	The path in <i>Filename</i> exceeds the maximum limit for the system.
<code>NotSupportedException</code>	<i>Filename</i> contains a colon (:) in the middle of the string.

### Format method—`String` class

```
String.Format(FormatString, Arg0, Arg1 [, ...])
```

*FormatString* is a string that contains text and/or formatting specifications. *Arg0* and *Arg1* are arguments that are to be formatted. The `[, ...]` notation indicates that more arguments may follow. The method returns a string that contains the data provided by the arguments *Arg0*, *Arg1*, and so on, formatted with the specifications found in *FormatString*.

Exception	Condition
ArgumentNullException	<i>FormatString</i> is set to Nothing.
FormatException	The format is invalid.

### FormatCurrency function

```
FormatCurrency(NumericExpression [,DecimalPoints [,IncludeLeading _
Digit [,UseParensForNegatives [,GroupDigits]]]])
```

Formats a number as currency, such as dollars and cents. The number will also include a dollar sign or other currency symbol. The value of *NumericExpression* is the number to be formatted. *DecimalPoints* is the number of decimal points the formatted number is rounded to. (By default the number is rounded to two places.) See the *FormatNumber* function for descriptions of the *IncludeLeadingDigit*, *UseParensForNegatives*, and *GroupDigits* arguments.

Exception	Condition
ArgumentException	The number of digits specified by <i>DecimalPoints</i> is greater than 99.
InvalidCastException	<i>NumericExpression</i> is nonnumeric.

### FormatDateTime function

```
FormatDateTime(DateExpression [, Format])
```

Formats a date expression. *DateExpression* is the date expression, such as the name of a Date variable, which is to be formatted. The second argument, *Format*, which is optional, is a value that tells Visual Basic how to format the date. It can be one of the following values: *DateFormat.GeneralDate*, *DateFormat.LongDate*, *DateFormat.ShortDate*, *DateFormat.LongTime*, or *DateFormat.ShortTime*.

Exception	Condition
ArgumentException	The format is invalid.

### FormatNumber function

```
FormatNumber(NumericExpression [,DecimalPoints [,IncludeLeadingDigit _
[,UseParensForNegatives [,GroupDigits]]]])
```

Returns a string containing a formatted number. The value of *NumericExpression* is the number to be formatted. *DecimalPoints* is the number of decimal points the formatted number is rounded to. (By default the number is rounded to two places.)

*IncludeLeadingDigit* can be one of the following values: *TriState.True* or *TriState.False*, or *TriState.UseDefault*. If *TriState.True* is passed, the number is formatted to include a leading zero if it is fractional. For example, the number .27 would be formatted as 0.27. If *TriState.False* is passed, no leading zero is used. If *TriState.UseDefault* is passed, or no argument is passed, the computer's regional settings are used.

*UseParensForNegatives* can also be one of the values `TriState.True` or `TriState.False`, or `TriState.UseDefault`. If `TriState.True` is passed as the argument, the formatted number is placed inside parentheses if it is negative. If `TriState.False` is passed as this argument, negative numbers are not placed inside parentheses. If `TriState.UseDefault` is passed, or no argument is passed, the function uses the regional settings.

*GroupDigits* can also be one of the values `TriState.True` or `TriState.False`, or `TriState.UseDefault`. If `TriState.True` is passed, or no argument is passed, the function inserts commas into the formatted number, according to the regional settings specifications. If `TriState.False` is passed as the argument, no commas are inserted.

Exception	Condition
<code>InvalidCastException</code>	<i>NumericExpression</i> is not numeric.

### FormatPercent function

```
FormatPercent(NumericExpression [,DecimalPoints _
               [,IncludeLeadingDigit [,UseParensForNegatives [,GroupDigits]]]])
```

Formats the numeric argument *NumericExpression* as a percent. It does this by multiplying the argument by 100, then rounding it two decimal places and adding a percent sign. *Decimal Points* is the number of decimal points the formatted number is rounded to. (By default the number is rounded to two places.) See the `FormatNumber` function for descriptions of the *IncludeLeadingDigit*, *UseParensForNegatives* and *GroupDigits* arguments.

Exception	Condition
<code>InvalidCastException</code>	<i>NumericExpression</i> is not numeric.

### GetItemChecked method—CheckedListBox control

```
CheckedListBox.GetItemChecked(Index)
```

`CheckedListBox` is the name of the `CheckedListBox` control. *Index* is the index of an item in the list. If the item is checked, the method returns `True`. Otherwise, it returns `False`.

Exception	Condition
<code>ArgumentException</code>	Index is less than zero or greater than or equal to the number of items in the list box.

### IndexOf method

```
StringExpression.IndexOf(SearchString)
```

```
StringExpression.IndexOf(SearchString, Start)
```

```
StringExpression.IndexOf(SearchString, Start, Count)
```

In the first format, *SearchString* is the string or character to search for within `StringExpression`. The method returns the character position, or index, of the first occurrence of *SearchString* if it is found within `StringExpression`. If *SearchString* is not found, the method returns `-1`.

Exception	Condition
<code>ArgumentNullException</code>	<i>SearchString</i> is Nothing.
<code>ArgumentOutOfRangeException</code>	<i>Start</i> is negative, <i>Count</i> is negative, <i>Start</i> is less than 0 or beyond the end of <i>StringExpression</i> , or <i>Count</i> plus <i>Start</i> is less than 0 or beyond the end of <i>StringExpression</i> .

### InputBox function

`InputBox(Prompt [, Title] [, Default] [, Xpos] [, Ypos])`

Displays an input box and returns the input typed by the user. The brackets are drawn around the *Title*, *Default*, *Xpos*, and *Ypos* arguments to indicate that they are optional. The first argument, *Prompt*, is a string that is displayed to the user in the input box. Normally, this string is a prompt requesting the user to enter a value. *Title* is a string that appears in the input box's title bar. If you do not provide a value for *Title*, the name of the project appears. *Default* is a string value that is initially displayed in the input box's text box. This value serves as the default input. If you do not provide a value for *Default*, the input box's text box will initially appear empty. *Xpos* and *Ypos* specify the input box's location on the screen. *Xpos* is an integer that specifies the distance of the input box's leftmost edge from the left edge of the screen. *Ypos* is an integer that specifies the distance of the topmost edge of the input box from the top of the screen. *Xpos* and *Ypos* are both measured in twips. (A twip is 1/1440th of an inch.) If *Xpos* is omitted, Visual Basic centers the input box horizontally on the screen. If *Ypos* is omitted, Visual Basic places the input box near the top of the screen, approximately one third the distance down.

### Insert method

`ListBox.Items.Insert(Index, Item)`

`ListBox` is the name of the `ListBox` control. *Index* is an integer argument that specifies the position where *Item* is to be placed in the `Items` property. *Item* is the item to add to the list.

Exception	Condition
<code>ArgumentOutOfRangeException</code>	<i>Index</i> is less than 0 or greater than or equal to the number of items in the list box.

### Int function

`Int(Number)`

Returns the integer portion of the argument *Number*.

Exception	Condition
<code>ArgumentNullException</code>	<i>Number</i> was not specified.
<code>ArgumentException</code>	<i>Number</i> is not numeric.

**IPmt function**

`IPmt(InterestRate, Period, NumPeriods, -Amount)`

Returns the required interest payment for a specific period on a loan. It assumes the loan has a fixed interest rate, with fixed periodic payments. *InterestRate* is the periodic interest rate. *Period* specifies which period you wish to calculate the interest payment for. (*Period* must be at least 1, and no more than the total number of periods of the loan.) *NumPeriods* is the number of periods of the loan. *Amount* is the loan amount (expressed as a negative number).

Exception	Condition
<code>ArgumentException</code>	<i>Period</i> is invalid.

**IsNumeric function**

`IsNumeric(StringExpression)`

Returns `True` if *StringExpression* contains a number, or `False` if the string's contents cannot be recognized as a number.

**Item method—System.Data.DataRow**

`ObjectVar.Item(ColumnName)`

*ObjectVar* is an object variable that references the new row. *ColumnName* is the name of the column. This expression returns the value stored in the column, or can be used in an assignment statement to store a value in the column.

Exception	Condition
<code>IndexOutOfRangeException</code>	The column cannot be found.
<code>InvalidCastException</code>	Occurs when setting a value whose data type does not match that of the column.
<code>DeletedRowInaccessibleException</code>	The row has been deleted.

**LCase function**

`LCase(StringExpression)`

Returns the lowercase equivalent of *StringExpression*.

**Left function**

`Left(StringExpression, Length)`

Extracts a specified number of characters from the left side of a string. The function returns a string that is the rightmost *Length* number of characters of *StringExpression*.

Exception	Condition
<code>ArgumentException</code>	<i>Length</i> is less than 0.

**Len function**

`Len(StringExpression)`

Returns the number of characters in *StringExpression*.

**Length method**

```
StringExpression.Length()
```

Returns the number of characters in a string.

**LTrim function**

```
LTrim(StringExpression)
```

Returns a copy of *StringExpression* with the leading spaces stripped away. The leading spaces are any spaces that pad the left side of the string.

**Mid function**

```
Mid(StringExpression, Start[, Length])
```

Extracts characters from the middle of *StringExpression* and returns them as a string. The second argument, *Start*, indicates the starting position of the string that is to be extracted. The third argument, *Length*, indicates the number of characters to extract (including the starting character). If *Length* is omitted, Mid will return all the characters from the starting position to the end of the string.

Exception	Condition
ArgumentException	Start is less than or equal to 0 or <i>Length</i> is less than 0.

**Now property**

```
Now
```

Returns a Date value containing the current date and time.

**NewRow method—System.Data.DataRow**

```
DataSet.Table.NewRow()
```

*ObjectVar* is the object variable, *DataSet* is the name of the dataset, *Table* is the name of the table. Returns the address of a new empty row that has the same column layout as the rows in *Table*.

**Open method—System.IO.File**

```
System.IO.File.Open(Filename, FileMode)
```

*Filename* is a string or a string variable specifying the path and/or name of the file to open. *FileMode* is a value specifying the mode in which the file will be opened. Valid modes are FileMode.Append, FileMode.Create, FileMode.CreateNew, FileMode.Open, FileMode.OpenOrCreate, and FileMode.Truncate. This method opens the file specified by *Filename* and returns the address of a FileStream object.

Exception	Condition
SecurityException	The calling application does not have the required permission.
ArgumentException	The filename or path is invalid.
ArgumentNullException	<i>Filename</i> is set to Nothing.
ArgumentOutOfRangeException	<i>FileMode</i> is invalid.
PathTooLongException	The path in <i>Filename</i> exceeds the maximum limit for the system.

*(continued)*

Exception	Condition
DirectoryNotFoundException	The directory specified does not exist.
IOException	The file already exists.
FileNotFoundException	The file specified does not exist.
NotSupportedException	<i>Filename</i> contains a colon (:) in the middle of the string.
UnauthorizedAccessException	<i>Filename</i> specifies a directory or a read-only file, or the operation is not supported.

**OpenText method—System.IO.File**

`System.IO.File.OpenText(Filename)`

*Filename* is a string or a string variable specifying the path and/or name of the file to open. This method opens the file specified by *Filename* and returns the address of a `StreamReader` object that may be used to write data to the file. If the file does not exist, a runtime error occurs.

Exception	Condition
SecurityException	The calling application does not have the required permission.
ArgumentException	The filename or path is invalid.
ArgumentNullException	<i>Filename</i> is set to <code>Nothing</code> .
PathTooLongException	The path in <i>Filename</i> exceeds the maximum limit for the system.
DirectoryNotFoundException	The directory specified does not exist.
FileNotFoundException	The file specified does not exist.
NotSupportedException	<i>Filename</i> contains a colon (:) in the middle of the string.

**Peek method—StreamReader class**

`ObjectVar.Peek()`

*ObjectVar* is an object variable referencing a `StreamReader` object. This method looks ahead in the file, without moving the current read position, and returns the very next character that will be read. If the current read position is at the end of the file (where there are no more characters to read), the method returns `-1`.

**Pmt function**

`Pmt(InterestRate, NumPeriods, -Amount)`

Returns the periodic payment amount for a loan. *InterestRate* is the periodic interest rate. *NumPeriods* is the number of periods of the loan. *Amount* is the loan amount (expressed as a negative number).

Exception	Condition
ArgumentException	<i>NumPeriods</i> is 0.

**PPmt function**

`PPmt(InterestRate, Period, NumPeriods, -Amount)`

Returns the principal payment for a specific period on a loan. It assumes the loan has a fixed interest rate, with fixed periodic payments. *InterestRate* is the periodic interest rate. *Period* specifies which period you wish to calculate the interest payment for. (*Period* must be at least 1, and no more than the total number of periods of the loan.) *NumPeriods* is the number of periods of the loan. *Amount* is the loan amount (expressed as a negative number).

Exception	Condition
<code>ArgumentException</code>	<i>Period</i> is invalid. It must specify a value greater than 1 and less than <i>NumPeriods</i> .

**Print method—PrintDocument control**

`PrintDocumentControl.Print()`

When the `Print` method is executed, it triggers a `PrintPage` event. It is in the `PrintPage` event procedure that you must write code to handle the actual printing.

Exception	Condition
<code>InvalidPrinterException</code>	The printer that is named in the <code>PrinterSettings.PrinterName</code> property does not exist.

**Randomize statement**

`Randomize [Number]`

Initializes the Visual Basic random number generator. The argument *Number* is any number or numeric expression you wish to use as a seed value.

**Read method—StreamReader class**

`ObjectVar.Read()`

*ObjectVar* is the name of a `StreamReader` object variable. The `Read` method reads only the next character from a file and returns the character code for that character. To convert the character code to a character, use the `Chr` function.

**ReadLine method—StreamReader class**

`ObjectVar.ReadLine()`

*ObjectVar* is the name of a `StreamReader` object variable. The method reads a line from the file associated with *ObjectVar* and returns the data that was read as a string.

Exception	Condition
<code>OutOfMemoryException</code>	There is not enough memory to hold the string that will be read.
<code>IOException</code>	An I/O error has occurred.

**ReadToEnd method—StreamReader class**

*ObjectVar*.ReadToEnd()

*ObjectVar* is the name of a `StreamReader` object variable. The `ReadToEnd` method reads and returns the entire contents of a file, beginning at the current read position.

Exception	Condition
<code>OutOfMemoryException</code>	There is not enough memory to hold the string that will be read.
<code>IOException</code>	An I/O error has occurred.

**Remove method—collections**

*Object*.Remove(*Expression*)

*Object* is the name of a collection. *Expression* can be either a numeric or string expression. If it is a numeric expression, it is used as an index value and the member at the specified index location is removed. If *Expression* is a string, the member with the key value that matches the string is removed.

Exception	Condition
<code>ArgumentException</code>	<i>Expression</i> is not specified, or specifies an index that is invalid.
<code>IndexOutOfRangeException</code>	<i>Expression</i> specifies an item that is not in the collection.

**Remove method—ListBox control**

*ListBox*.Items.Remove(*Item*)

*ListBox* is the name of the `ListBox` control. *Item* is the item you wish to remove.

**RemoveAt method—CurrencyManager**

*ObjectVar*.RemoveAt(*Index*)

Removes a row in the dataset. *ObjectVar* is an object variable that references the `CurrencyManager` object, and *Index* is the position of the row that is to be removed.

Exception	Condition
<code>IndexOutOfRangeException</code>	<i>Index</i> specifies a row that does not exist.

**RemoveAt method—ListBox control**

*ListBox*.Items.RemoveAt(*Index*)

*ListBox* is the name of the `ListBox` control. *Index* is the index of the item to be removed.

Exception	Condition
<code>ArgumentOutOfRangeException</code>	<i>Index</i> is less than 0 or greater than or equal to the number of items in the list box.

**Right function**

```
Right(StringExpression, Length)
```

Extracts a specified number of characters from the right side of a string. The function returns a string that is the rightmost *Length* number of characters of *StringExpression*.

Exception	Condition
ArgumentException	<i>Length</i> is less than 0.

**Rnd statement**

```
Rnd [(Number)]
```

Returns a single precision random number between 0.0 and 1.0. The argument, *Number* is shown in brackets, indicating that it is optional. If *Number* is less than 0, it is used as a seed to generate a new sequence of numbers. If *Number* is 0, Rnd returns the last random number generated. If *Number* is greater than 0, Rnd returns the next random number in the sequence. If *Number* is omitted, Rnd returns the next random number in the sequence.

**RTrim function**

```
RTrim(StringExpression)
```

Returns a copy of *StringExpression* with the trailing spaces stripped away. The trailing spaces are any spaces that pad the right side of the string.

**SetDataObject—Clipboard**

```
Clipboard.SetDataObject(Text)
```

```
Clipboard.SetDataObject(Text, Remain)
```

The argument, *Text*, is the text you wish to store in the clipboard. In the second format, the argument *Remain* is a Boolean value. If *Remain* is True, the text will remain in the clipboard after the application exits. If *Remain* is False, the text will not remain in the clipboard after the application exits.

Exception	Condition
ExternalException	The data could not be retrieved from the clipboard.
ThreadStateException	The ApartmentState property of the application is not set to ApartmentState.STA .
ArgumentNullException	The value of <i>Text</i> is Nothing.

**Show method—Form class**

```
ObjectVariable.Show()
```

*ObjectVariable* is the name of an object variable that references an instance of a form. This method displays the form in modeless style.

**Show method—MessageBox class**

```
MessageBox.Show(Message)
```

```
MessageBox.Show(Message, Caption)
```

```
MessageBox.Show(Message, Caption, Buttons)
```

```
MessageBox.Show(Message, Caption, Buttons, Icon)
```

```
MessageBox.Show(Message, Caption, Buttons, Icon, DefaultButton)
```

When `MessageBox.Show` is executed, a message box (which is a Windows dialog box) appears on the screen. In the first format shown, *Message* is a string that is displayed in the message box. In the second format, *Caption* is a string to be displayed in the message box's title bar. In the third format, *Buttons* is a value that specifies which buttons to display in the message box. In the fourth format, *Icon* is a value that specifies an icon to display in the message box. In the fifth format, the *DefaultButton* argument is a value that specifies which button to select as the default button. The default button is the button that is clicked when the user presses the `[Enter]` key.

Exception	Condition
<code>InvalidEnumArgumentException</code>	The <i>Buttons</i> , <i>Icon</i> , or <i>DefaultButton</i> value is invalid.
<code>InvalidOperationException</code>	The call was made from a process that was not running in user interactive mode.

### ShowDialog method—Form class

```
ObjectVariable.ShowDialog()
```

*ObjectVariable* is the name of an object variable that references an instance of a form. This method displays the form in modal style.

### Sort method—Array class

```
Array.Sort(ArrayName)
```

Sorts the contents of an array in ascending order. *ArrayName* is the name of the array you wish to sort.

Exception	Condition
<code>ArgumentNullException</code>	The array is set to <code>Nothing</code> .
<code>RankException</code>	The array is multidimensional.
<code>InvalidOperationException</code>	An exception was thrown by the comparer.

### Today property

```
Today
```

Returns a `Date` value containing the current date.

### ToLower method

```
StringExpression.ToLower()
```

Returns the lowercase equivalent of a string expression.

### ToString method

```
VariableName.ToString()
```

Returns a string representation of the calling variable or the object referenced by the variable.

**ToUpper method**

```
StringExpression.ToUpper()
```

Returns the uppercase equivalent of a string expression.

**Trim function**

```
Trim(StringExpression)
```

Returns a copy of *StringExpression* with the leading and trailing spaces stripped away. The leading spaces are any spaces that pad the left side of the string, and the trailing spaces are any spaces that pad the right side of the string.

**Trim method**

```
StringExpression.Trim()
```

Returns a copy of the string expression with all leading and trailing spaces removed.

**TrimEnd method**

```
StringExpression.TrimEnd()
```

Returns a copy of the string expression with all trailing spaces removed.

**TrimStart method**

```
StringExpression.TrimStart()
```

Returns a copy of the string expression with all leading spaces removed.

**UCase function**

```
UCase(StringExpression)
```

Returns the uppercase equivalent of *StringExpression*.

**Update method—DataAdapter**

```
DataAdapter.Update(DataSet)
```

*DataAdapter* is the name of a data adapter and *DataSet* is the name of a data set. After this statement executes, the changes that have been made to the data set will be saved to the data adapter's data source.

Exception	Condition
SystemException	The source table could not be found.
DBConcurrencyException	The update method attempted an INSERT, UPDATE, or DELETE statement, which affected no records.

**Val function**

```
Val(StringExpression)
```

Returns the numeric value of the *StringExpression* as a Double.

Exception	Condition
OverflowException	The value is too large.
InvalidCastException	The number is badly formed.

*(continued)*

Exception	Condition
ArgumentException	The value passed as the argument cannot be converted to a number.

**Write method—BinaryWriter class***ObjectVar.Write(Data)*

*ObjectVar* is the name of a `BinaryWriter` object variable. *Data* is the data that is to be written to the file. *Data* can be a constant or the name of a variable.

Exception	Condition
IOException	An I/O error has occurred.

**Write method—StreamWriter class***ObjectVar.Write(Data)*

*ObjectVar* is the name of a `StreamWriter` object variable. *Data* is the data that is to be written to the file. *Data* can be a constant or the name of a variable. This method can be used to write data to a file without terminating the line with a newline character.

Exception	Condition
IOException	An I/O error has occurred.
ObjectDisposedException	The <code>StreamWriter</code> object's buffer is full, or <code>AutoFlush</code> is <code>True</code> , and the <code>StreamWriter</code> object is closed.
NotSupportedException	The <code>StreamWriter</code> object's buffer is full, or <code>AutoFlush</code> is <code>True</code> , and the contents of the buffer cannot be written because the stream is fixed-size and the object is at the end of the stream.

**WriteLine method—Debug***Debug.WriteLine(Output)*

*Output* is a constant or a variable whose value is to be displayed in the *Output* window.

**WriteLine method—StreamWriter class***ObjectVar.WriteLine(Data)*

*ObjectVar* is the name of a `StreamWriter` object variable. *Data* is the data that is to be written to the file. *Data* can be a constant or the name of a variable. The method writes the data to the file and then writes a newline character immediately after the data.

Exception	Condition
IOException	An I/O error has occurred.