Rocket | BlueZone®

# BlueZone Advanced Automation

## Developer's Guide

*Version 6.1*

January 2013
BZAA-0601-DG-02

Rocket®

# Notices

## Edition

**Publication date:** January 2013

**Book number:** BZAA-0601-DG-02

**Product version:** BlueZone Advanced Automation Version 6.1

## Copyright



## Trademarks



## Examples

This information might contain examples of data and reports. The examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## License agreement



**Note**

This product may contain encryption technology. Many countries prohibit or restrict the use, import, or export of encryption technologies, and current use, import, and export regulations should be followed when exporting this product.

## Contact information

Website: www.rocketsoftware.com

Rocket Software, Inc. Headquarters
77 4th Avenue, Suite 100
Waltham, MA 02451–1468
USA
Tel: +1 781 577 4321
Fax: +1 617 630 7100

# Contacting Global Technical Support

If you have current support and maintenance agreements with Rocket Software, you can access the Rocket Customer Portal and report a problem, download an update, or read answers to FAQs. The Rocket Customer Portal is the primary method of obtaining support.

To log in to the Rocket Customer Portal, go to:

www.rocketsoftware.com/support

If you do not already have a Rocket Customer Portal account, you can request one by clicking **Need an account?** on the Rocket Customer Portal login page.

Alternatively, you can contact Global Technical Support by email at support@rocketsoftware.com.

# Contents

# Help options

BlueZone products offer two ways to access help information: a locally installed `.chm` file or Web-based HTML help.

By default, BlueZone Desktop is configured to use the locally installed help file. If you want to use Web help, you must edit the `global.ini` file:

1. Open `global.ini` in a text editor.
2. In the `[Help]` section, change `UseWebHelp=No` to `UseWebHelp=Yes`.
3. Save and close the file.
4. Run `setup.exe` to reinstall BlueZone.

**Note**

BlueZone will continue to use the locally installed help file until you run `setup.exe`.

# What's new in this release

This section summarizes the significant improvements or enhancements for BlueZone Advanced Automation Version 6.1 and refers you to relevant sections of this book for more information. Minor modifications to the text are not listed.

**BZAA-0601-DG-02** January 2013

- The VT send keys support friendly syntax. Refer to VT send keys, on page 253 for more information.

**BZAA-0601-DG-01** September 2012

- The method to configure the help options have changed. Refer to Help options, on page 12 for more information.
- `TelnetEncryption` is a new Host Automation Object property. Refer to TelnetEncryption, on page 78 for more information.
- The SendKey method can be used in UTS displays. Refer to SendKey, on page 73 and UTS send keys, on page 257 for more information.
- The `WaitForKeys` method `TimeoutVal` parameter value has changed from milliseconds to seconds. The maximum finite wait time is 65535 seconds, approximately 18 hours. Refer to WaitForKeys, on page 81 for more information.
- You can now run multiple versions of BlueZone side-by-side. Refer to Multiple side-by-side versions of BlueZone, on page 96 for more information.
- The BlueZone LIPI COM Object is new. Refer to BlueZone LIPI COM Object, on page 202 for more information.

# Chapter 1: Introduction

Welcome to the *BlueZone Advanced Automation Developer's Guide*. This purpose of this guide is to provide the professional application developer with the tools necessary to incorporate BlueZone into advanced automation applications.

The BlueZone family of terminal emulation clients provide support for a wide range of scripting and automation options. BlueZone scripting and automation capabilities range from the simple recording of keystrokes, to object oriented programming, to the integration of BlueZone into VBA applications.

BlueZone proprietary scripting is targeted at the end user level and is not considered to be Advanced Automation and as such are not discussed in this guide. Only BlueZone Advanced Automation options are discussed in the this guide.

## BlueZone Advanced Automation options

The following options are provided to aid in the development scripts and automation solutions.

**BlueZone Script Host**
A language-independent host for ActiveX scripting engines on 32-bit Windows platforms.

**BlueZone Basic**
A Visual Basic for Applications (VBA) and VBScript compatible Basic Scripting Language which can be used to add functionality to the BlueZone family of terminal emulation clients or Web pages to automate complex tasks.

**BlueZone Host Automation Object**
A Component Object Model (COM) software component for 32-bit Windows platforms.

**BlueZone Plus VBA (Visual Basic for Applications)**
A powerful development technology for rapidly customizing Windows applications like BlueZone and integrating them with existing data and systems.

**Visual Basic Application Support**
The BlueZone terminal emulation client can be embedded into a VBA form to provide host information to the Visual Basic application.

For help with BlueZone Macro Keystroke Recorder and BlueZone Proprietary Scripting, refer to *BlueZone Display and Printer Help*.

## BlueZone Advanced Automation tools

The following automation tools are provided to aid in the development of BlueZone script and automation solutions.

**BlueZone Script Host and Debugger**
An IDE that is used to create and edit BlueZone scripts.

**BlueZone Dialog Editor**
An IDE that is used to create dialogs that can be inserted into BlueZone scripts.

# Migrating from another emulator to BlueZone

Migrating from another emulator to BlueZone is greatly simplified by using the BlueZone Migration Toolkit.

The typical BlueZone customer has thousands of desktops. Converting these to BlueZone is best accomplished using the BlueZone migration tools and techniques developed over the past several years. The migration tools take you from discovering who is using the existing emulator and how it is being used, to developing a conversion, migration, and deployment plan.

The toolkit consists of conversion utilities, compatible APIs, runtime files, and migration planning documents to facilitate the conversion to BlueZone.

The following are examples of items that can be converted using the Migration Toolkit:

- Configuration files
- Scripts
- Macros
- Microsoft Office Automation
- Visual Basic for Applications
- Visual Studio Projects (VB, .NET, C/C++, and C#)

# BlueZone OLE automation

OLE (Object Linking and Embedding) is a Microsoft Windows standard for communications between applications. BlueZone can be linked to Windows applications like Microsoft Word and Microsoft Excel that support OLE.

# Chapter 2: BlueZone Script Host and Debugger

The BlueZone Script Host and Debugger is a language-independent host for ActiveX scripting engines on 32-bit Windows platforms. This tool allows you to run BlueZone Basic, Visual Basic Scripting Edition (VBScript), and JScript natively within the base operating system; either on Windows 95, Windows 98, Windows NT, Windows 2000, or Windows XP; and acts as a host for other ActiveX-supported scripting languages such as Perl, Rexx, or Python. In addition, BlueZone Script Host and Debugger allows scripts to communicate with BlueZone emulation software products. Using the scripting languages you already know, you can now write scripts to execute common tasks on a wide variety of host systems, automate user input, obtain data from host systems, initiate file transfers, and more.

Using BlueZone Script Host and Debugger, BlueZone can record and playback scripts using BlueZone Basic, VBScript, or JavaScript. Once recorded, these scripts can be played back as is or edited using the BlueZone Script Host and Debugger. The record and playback feature makes using BlueZone Basic, VBScript, and JavaScript available to the non-technical user.

The advantages of BlueZone Script Host and Debugger are:

- Very powerful
- Can control multiple host sessions simultaneously
- Use of industry standard scripting languages
- Direct access to read from and write to the host screen
- File I/O
- Variable support
- COM compliance allows any other COM complaint component to be loaded by the script to extend its functionality
- Powerful editing and debugging features
- Ability to view the value of script variables while executing/debugging the script
- Dialog support to create Windows dialogs for user interaction

## Overview

BlueZone Script Host and Debugger is a Component Object Model (COM) scripting host application that utilizes the capabilities of the BlueZone Dynamic Data Exchange (DDE) and High-Level Language API (HLLAPI) interfaces. In addition to the scripting engine's properties and methods, BlueZone Script Host and Debugger adds methods that enable scripts to interact with the BlueZone emulation sessions and the host system.

BlueZone Script Host and Debugger can be used to debug scripts, pause script execution, step through statements and subroutines, and evaluate expressions. The process of debugging scripts is a simple one and begins with the setting of one or more debug breakpoints.

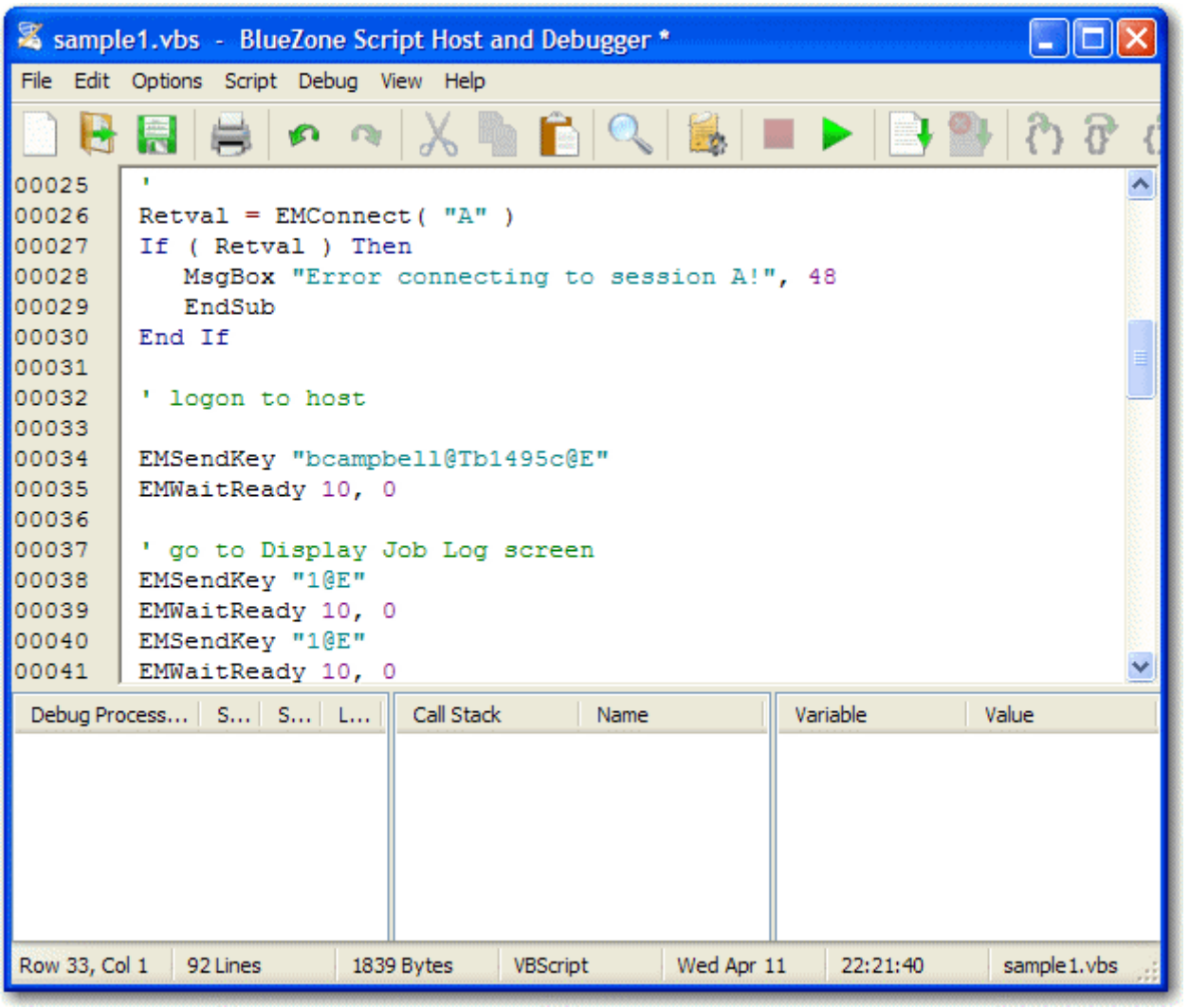
**Note**

The Microsoft Machine Debug Manager must be installed on your workstation to use the debugging features of the BlueZone Script Host and Debugger. If you do not have the Machine Debug Manager installed, BlueZone Script Host and Debugger detects this condition and automatically downloads it.

### BlueZone Script Host and Debugger application

The following is an example of editing a script in the BlueZone Script Host and Debugger:

Figure 1: Script Host and Debugger interface




## BlueZone Dialog Editor application

BlueZone Script Host and Debugger is designed to work with the BlueZone Dialog Editor. The BlueZone Dialog Editor is a standalone application that is used to create feature rich dialogs that can be incorporated into BlueZone scripts.

Refer to BlueZone Dialog Editor, on page 42 for more information on using the Dialog Editor.

# Use the BlueZone Script Host and Debugger

Many of the BlueZone Script Host and Debugger features, like **File** ® **Open**, work exactly the same way as any standard Windows program. This guide is concerned only with those features that are unique to the BlueZone Script Host and Debugger.

# Options menu

Use **Options** menu to select the scripting language that you are working in. Also, you can select **Custom Script Engine** and **Script Host Properties** from this menu.

**Custom Script Engine**

**Script Host Properties**

Script Host Properties is used to change the font used and to access the Speed Control. It contains the following tabs:

**General tab**

The General tab is used to select the **Script Host Type**.

**Security tab**

The Security tab is used to enable script encryption and password authentication. These two features work together and must be turned on or off together.

To turn on this feature, check the **Enable Script File Encryption and Password Authentication** check box.

You can also choose to enable the **Show Password on Status Line When Editing Scripts** check box. This is helpful when you are creating and editing many scripts so you don't forget the password.

Refer to Script encryption and password protection, on page 19 for more information.

**Font tab**

The Font tab is used to change the font used by the application.

Refer to Changing the default font, on page 21 for more information.

**Speed tab**

The Speed tab is used to control the amount of wait time that is associated with the `EMWaitReady` and `EMWaitCursor` functions.

Refer to Using the speed control, on page 21 for more information.

# Setting debug breakpoints

Debug breakpoints can be added and removed in two ways:

- Click the desired line number to toggle.
- Place the cursor in the desired line and click **Debug** ® **Toggle Breakpoint** from the menu bar or click the Toggle Breakpoint toolbar icon  .

When a breakpoint is set, the line number is highlighted in red and a red dot appears to the left of the line number.

After adding one or more breakpoints to the script, the Debug menu bar and toolbar buttons can be used to control script execution while analyzing the script.

# Debug menu options and toolbar icons

The following toolbar icons can also be accessed through the **Debug** menu.

Table 1: Debugger icons

| Icon | Name | Description |
|---|---|---|
|  | Go | Starts or continues script debugging by executing until the next Debug Breakpoint is hit. |
|  | Stop Debugging | Stops script debugging. |
|  | Step Into | Steps into subroutine, function or procedure. Script execution will halt on the next statement within the subroutine, function, or procedure. |
|  | Step Over | Steps over subroutine, function, or procedure. Script execution will halt on the next statement following the subroutine, function, or procedure. |
|  | Step Out | Steps out of subroutine, function, or procedure. Script execution will halt on the next statement following the subroutine, function, or procedure. |
|  | Run to Cursor | Executes script until the line number where the caret resides is hit. |
|  | Toggle Breakpoints | Adds or removes a Debug Breakpoint on the line where the caret resides. |
|  | Clear All Breakpoints | Clears all Debug Breakpoints. |
|  | Evaluate | Invokes the Evaluate Expression dialog. Enter an expression to evaluate then click **Evaluate**. |

## Debug panes

The debug panes are used to provide script code and data information while debugging. The debug panes consist of three windows:

- Debug Process/Thread window: Displays a list of processes and threads involved in debugging.
- Call Stack window: Displays a list of nested subroutines, functions and procedures showing a path of execution to the current call stack.
- Variable window: Displays a list of local variables of the current call stack and their values.

The debug panes can be shown or hidden by clicking **View** ® **Debug Panes** from the menu bar or the **Debug Panes** toolbar icon  .

# Script encryption and password protection

BlueZone Script Host and Debugger has an optional security feature that allows the encryption and password protection of scripts to prevent unauthorized users from running, viewing, or editing them. The script encryption feature is toggled on and off using a check box in the **Security** tab of the **Options** ® **Script Host Properties** dialog.

When enabled, a lock appears on the status bar, located at the bottom of the display, to indicate that any scripts saved are saved with encryption and password protection.

When disabled, scripts are saved in clear text that is readable with any text editor, like Notepad.

### Encrypting scripts with password protection

1. Launch BlueZone Script Host and Debugger.
2. From the menu bar, click **Options** ® **Scripting Host Properties**.
3. Click the **Security** tab.
4. Enable the **Script File Encryption and Password Authentication** check box.

   **Note**

   As an option, you can choose to have the password displayed on the Status Line while you are editing the script.

5. Open or create a script using the Script Host editor.
6. Click **File** ® **Save** or **File** ® **Save As**.

   The first time this is performed on a script file, you are prompted to type a password.
7. The file is now saved and encrypted. To play the script, the end user must know the password and type it each time a script is played.

### Opening encrypted script files for editing or playing

1. From the Script Host and Debugger menu bar, click **File** ® **Open**.
2. At the prompt, type the password to decrypt the script file.
   The script displays in the debugger main window and can be edited or played.

### Playing encrypted script files from BlueZone

1. In a BlueZone display emulator, click **Script** ® **Play** from the menu bar.

   The Play Script dialog appears.
2. Select the .VBS file type and highlight the script to play.
3. Click **Open**.

   The end user is prompted to type the password to play the script.
4. Type the password and the script plays.

**CAUTION**

Script passwords cannot be recovered. If the password is lost or forgotten, the script cannot be opened and must be rewritten. It is advisable to create clear-text backups of each encrypted script.

## Configuring the Custom Script Engine

1. From the Script Host and Debugger menu bar, click **Options** ® **Custom Script Engine**.

   **Note**

   In order to support a custom scripting engine, you must know either the **Program Identifier** or the **Class Identifier** of the scripting engine. Consult the documentation for the particular scripting engine you want to use.

2. Type the **Program Identifier** or the **Class Identifier** in the corresponding field.

   If you are using the **Class Identifier**, type a description in the **Program Identifier** field to display on the application status line.
3. Type the **Script Filename Extension** that is normally used with this scripting engine.
4. Click **OK**.

# Changing the default font

1. From the Script Host and Debugger menu bar, click **Options** ® **Script Host Properties**.

   The Script Host Properties dialog opens.
2. Click the **Font** tab.

   The Font Selection dialog opens.
3. Click **Change**.
   A standard Windows font selection dialog opens.

   **Note**

   Only fixed width fonts are displayed.
4. Select the font that you want to use and click **OK**.
5. Click **OK** again to exit the Script Host Properties dialog.

# Using the speed control

The script speed control works in conjunction with two BlueZone Script methods:

- EMWaitReady
- EMWaitCursor

These methods have a speed control option that allows you to use the **Speed Control Slider** in order to control the amount of delay that the script waits when these methods are used.

This feature is designed to make it easier to refine your script and achieve the optimal script playback speed.

To take advantage of this feature, you must use the proper parameter value setting in order for the BlueZone Script Host to know that you want to use the script speed control to control the script delay.

In the `EMWaitReady` method, there is a parameter called `ExtraWaitVal`. This parameter is used to set a specific number of screen writes, or a specific delay value for the script to wait. To use the script speed control, use a `-1` for the `ExtraWaitVal` value.

For example:

```
EMWriteScreen "list", 19, 7

EMSendKey "@E"

EMWaitReady -1 'uses the Speed Control to determine script delay

EMWriteScreen "x", 4, 16
```

**Note**

The script speed control for `EMWaitCursor` works in a similar fashion.

To access and change the speed control:

1. Click **Options** ® **Script Host Properties** from the menu bar.

   The Script Host Properties dialog opens.
2. Click the **Speed** tab.
3. Drag the **Speed Control Slider** from left to right as needed.
4. Click **OK** to set the speed and exit the dialog.

# Script Host methods

The BlueZone Script Host methods can be incorporated into scripts just like any other scripting command. Select a method to view details about functionality, parameters, and return codes. All of the examples are written in VBScript.

```
Val CloseSession( SessionTypeVal, SessionIdentifierVal );

Val EMConnect( SessionShortNameStr );

Val EMFocus( );

Val EMGetCursor( RowVal, ColumnVal );

Val EMPrintScreen;

Val EMReadScreen( BufferStr, LengthVal, RowVal, ColumnVal );

Val EMReceiveFile( ReceiveStr ); *

Val EMSearch( SearchStr, RowVal, ColumnVal );

Val EMSendFile( SendStr ); *

Val EMSendKey( KeyStr );

Val EMSetCursor( RowVal, ColumnVal );

Val EMWaitCursor( TimeoutVal, RowVal, ColumnVal, ExtraWaitVal );

Val EMWaitForText( TextStr, RowVal, ColumnVal, TimeOutVal );

Val EMWaitReady( TimeoutVal, ExtraWaitVal );

Val EMWriteScreen( WriteStr, RowVal, ColumnVal );

Str InputBox( PromptStr, TitleStr, DefaultStr );

Val MsgBox( MessageStr, FlagsVal );

Val OpenSession( SessionTypeVal, SessionIdentifierVal, ConfigFileStr,
TimeoutVal, WaitPaintsVal );

Val Pause( PauseTimeVal );

Val PrintString( PrintStr, FlagVal );

Val Run( CommandStr );

Val StopScript;

Str Str( ValueVal );

Val Val( StringStr );
```

* Not supported in BlueZone VT

## EMConnect

Opens a conversation with the BlueZone Display session. The `EMConnect` command must be called before any other BlueZone Script Host methods that access data in the host screen.

`EMConnect` auto-connects to the BlueZone session that launched Script Host or it searches for the first available session if launched from BlueZone desktop when no short name session identifier is specified.

**Parameters**

**`SessionShortName`**
Uniquely identifies the BlueZone Display session. The Session Name corresponds to the HLLAPI Short Name Session Identifier configured in the **Options** ® **API settings** in BlueZone Display.

If the `SessionShortName` parameter is omitted, you must include empty quotes (`EMConnect""`) to launch the first available session. If you do not include the empty quotes, you receive an error.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The script can only be connected to one display session at a time. When connecting to multiple sessions, any previously connected session is automatically disconnected.

**Example**

```
UserId = "John"
Password = "Smith"

ResultCode = EMConnect( "A" )

If ( ResultCode = 0 ) Then
   EMWriteScreen Userid, 5, 63
   EMWriteScreen Password, 6, 63
   SendKey "@E"
   EMWaitReady 10, 1
End If
```

# CloseSession

Disconnects from the host system and closes the BlueZone Display session window.

**Parameters**

**`SessionTypeVal`**
0 - Mainframe; 1 - iSeries; 2 - VT

**`SessionIdentifierVal`**
1 for S1; 2 for S2; 3 for S3; etc.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The `CloseSession` function can be used to end a session started with the `OpenSession` function.

Starting with BlueZone version 3.3, the `CloseSession` method works with BlueZone Web-to-Host when using the embedded client mode.

**Note**

The embedded BlueZone session must first be connected to the program or script by using the `Connect` method.

**Example**

```
OpenSession 0, 2, "System2.Zmd", 30, 1

ResultCode = EMConnect( "A" )

If ( ResultCode = 0 ) Then
      ' interact with host
End If

CloseSession 0, 1
```

# EMFocus

Brings the BlueZone Display session window into the foreground.

**Parameters**

None.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The script must first be connected to the session using the `EMConnect` method. The `EMFocus` method activates and directs all keyboard input to the BlueZone session window.

**Example**

```
EMConnect "A"
EMFocus
```

# EMGetCursor

Retrieves the host screen cursor position.

**Parameters**

**`RowVal`**

Variable to contain cursor's row position.

**`ColumnVal`**

Variable to contain cursor's column position.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

When the `EMGetCursor` function returns, the `RowVal` and `ColumnVal` variables contain the host screen cursor position.

**Example**

```
EMConnect "A"
EMGetCursor row, col
MsgBox "Cursor is at row " & row & ", column " & col, 0
```

# EMPrintScreen

Performs a BlueZone Print Screen function.

**Parameters**

None.

**Example**

```
EMConnect "A"
EMPrintScreen
```

This method can also be used send a text string which adds a prefix to the header page.

**Optional parameters**

"Text String"

**Example**

```
EMConnect "A"
EMPrintScreen "Pre-pended Header Text"
```

In the above example, the text string "Pre-pended Header Text" appears on the top of the printed page before any configured page header.

# EMReadScreen

Retrieves data from the host screen.

**Parameters**

**BufferStr**
Variable to contain host screen data.

**LengthVal**
Number of characters to read.

**RowVal**
Row position.

**ColumnVal**
Column position.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

When the EMReadScreen function returns, the BufferStr variable contains the host screen data.

**Example**

```
EMConnect "A"
EMReadScreen Buf1, 10, 7, 16
MsgBox "Buf1 is " & Buf1, 0
```

# EMReceiveFile

Initiates a IND$FILE file transfer download from the mainframe host system.

**Parameters**

**`ReceiveStr`**
IND$FILE Receive command string.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
Script execution is suspended until the file transfer is complete. The `ReceiveStr` command parameters correspond to what you would normally enter at the DOS prompt.

The example below receives the file `SLSREPRT` from the host system to the PC file `C:\BlueZone\Transfer\Sales.Rpt` through a BlueZone session having the short name session ID of `A`.

**Example**

```
EMReceiveFile "C:\BlueZone\Transfer\Sales.Rpt A:SLSREPRT (ASCII CRLF)"
```

# EMSearch

Searches the host screen for some specified text.

**Parameters**

**`SearchStr`**
Specifies the text to search for.

**`RowVal`**
On input, the variable containing the row position where the search is to begin. On output, this variable contains the row position where the text was found.

**`ColumnVal`**
On input, the variable containing the column position where the search is to begin. On output, this variable contains the column position where the text was found.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
When the `EMSearch` function returns, the `RowVal` and `ColumnVal` variables contain the position where the text was found. If the text was not found, or if the function returned an error, the `RowVal` and `ColumnVal` variables contain zero.

**Example**

```
EMConnect "A"
row = 1
col = 1
EMSearch ">>", row, col
MsgBox ">> is at position " & row & ", " & col, 0
```

# EMSendFile

Initiates a IND$FILE file transfer upload to the mainframe host system.

**Parameters**

**SendStr**
IND$FILE Send command string.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
Script execution is suspended until the file transfer is complete. The SendStr command parameters correspond to what you would normally enter at the DOS prompt.

The example below sends the file SALES.RPT from your PC to the CMS file SLSREPRT on the host system via a BlueZone session having the short name session ID of A.

**Example**

```
EMSendFile "C:\BlueZone\Transfer\Sales.Rpt A:SLSREPRT (ASCII CRLF"
```

# EMSendKey

Sends a sequence of keys to the display session.

**Parameters**

**KeyStr**
String of key codes. See the Remarks section for a complete listing of valid key codes and descriptions.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The EMSendKey function affects the host screen as if the user were typing on the keyboard.

Refer to the following tables for a list of host function keys and their corresponding codes. If a character is used in the code, then the case of the character is important.

- IBM 3270 / 5250 Send Key Table, on page 251
- BlueZone VT Send Key Table, on page 253

**Note**

If you want to use the at sign (@) in the data string, you must use the two-byte code "@@".

**Example 1**
This example utilizes standard mnemonic syntax. For example, @T for TAB and @E for ENTER.

```
EMSendKey "John@TSmith@E"
EMWaitReady 10, 1
```

**Example 2**
This example utilizes friendly syntax. For example, <Tab> for TAB and <Enter> for Enter.

```
EMSendKey "John<Tab>Smith<Enter>"
EMWaitReady 10, 1
```

**Note**

If you are converting macros or scripts to BlueZone and your existing macros or scripts contain the friendly syntax, there is no need to convert them to mnemonic syntax.

# EMSetCursor

Sets the host screen cursor position.

**Parameters**

**RowVal**
Row position.

**ColumnVal**
Column position.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
BlueZone VT attempts to move the cursor on the screen by sending cursor movement commands to the host. Not all VT applications/screens support cursor movement commands.

**Example**

```
EMConnect "A"
EMSetCursor 6, 53
EMSendKey "myuserid@Tmypassword@E"
EMWaitReady 10, 1
```

# EMWaitCursor

Suspends script execution until the host screen is ready for keyboard input and the cursor is at the specified location.

**Parameters**

**TimeoutValue**
The number of seconds to wait before returning with a `session is busy` error code.

**RowValue**
Specifies the cursor row position in the host screen.

**ColumnValue**
Specifies the cursor column position is the host screen.

**ExtraWaitVal**
The number of milliseconds to validate for a keyboard unlocked status.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The `EMWaitCursor` function can be used to verify that a specific host screen is being displayed before continuing with script execution.

If `ExtraWaitVal` is a value in the range of 1, 2, 3, … ,50 then script execution is suspended until the host machine sends the specified number of screen writes containing the keyboard restore/unlock command.

If `ExtraWaitVal` is greater than 50, the script execution is suspended until the specified number of milliseconds have transpired.

**Example 1**

```
EMWriteScreen "list", 19, 7
EMSendKey "@E"
ResultCode = EMWaitCursor( 10, 4, 16, 3 ) 'wait for 3 keyboard restores
If ( ResultCode != 0 ) Then
   MsgBox "Timeout Error in Script!", 48
End If
```

**Example 2 (using speed control)**

```
EMWriteScreen "list", 19, 7
EMSendKey "@E"
ResultCode = EMWaitCursor( -1 ) 'use Speed Control to determine script delay
If ( ResultCode != 0 ) Then
   MsgBox "Timeout Error in Script!", 48
End If
```

# EMWaitForText

Suspends script execution until the desired text is found in the host screen.

**Parameters**

**`TextStr`**
The text string that you want to search for in the host screen.

**`RowValue`**
Specifies the start row position in the host screen where the search is to begin.

**`ColumnValue`**
Specifies the start column position in the host screen where the search is to begin.

**`TimeoutValue`**
The number of seconds to wait before returning with a `session is busy` error code.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The `EMWaitForText` function can be used to verify that a specific host screen is being displayed before continuing with script execution.

**Example**

```
EMConnect "A"
ResultCode = EMWaitForText( "S103XWRM", 1, 1, 10 ) 'wait for text
If ( ResultCode != 0 ) Then
   MsgBox "Error " & ResultCode & " waiting for text.", 48
End If
```

# EMWaitReady

Suspends script execution until the host screen is ready for keyboard input.

### Parameters

**`TimeoutVal`**
The number of seconds to wait before returning with a `session is busy` error code.

**`ExtraWaitVal`**
The number of milliseconds to validate for a keyboard unlocked status. For a detailed explanation, see the Remarks section below.

The above features behave differently when scripting non-IBM hosts. See Non-IBM Remarks below.

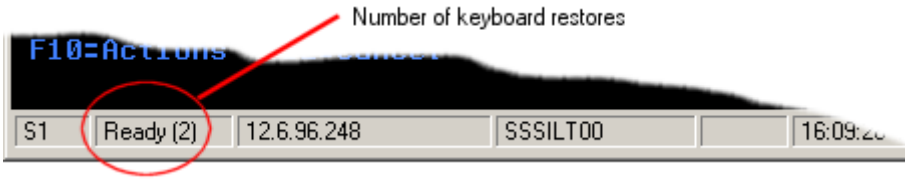
### Returns

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

### Remarks

The `EMWaitReady` function must be called each time after sending an attention identifier key (such as a PF key) to the display session.

If `ExtraWaitVal` is a value in the range of 1, 2, 3, on up to 50, then script execution will be suspended until the host machine sends the specified number of keyboard restores. Refer to the BlueZone status bar to determine the keyboard restore count for a given screen. In the following screen shot **Ready (2)** on the status bar means two keyboard restores were detected when this particular screen was written by the host.




If `ExtraWaitVal` is 51 or higher, the operation of the parameter changes to specify the number of milliseconds to wait after the keyboard lock has been detected prior to executing the next script command.

**Note**

If `-1` is specified for the `ExtraWaitVal` parameter, then the speed control setting in **Options** ® **Script Host Properties** is used.

### Non-IBM remarks

`TimeoutVal` and `ExtraWaitVal` behave differently when scripting non-IBM hosts. That is because keyboard locked status is not supported on non-IBM hosts. When scripting on non-IBM hosts, set `TimeoutVal` to `0` and treat `ExtraWaitVal` as a pause before the scripts moves on to the next command. See Example 3 below.

### Example 1

```
EMWriteScreen "list", 19, 7
EMSendKey "@E"
EMWaitReady 10, 2000  'wait 2 seconds for next screen to come down
EMWriteScreen "x", 4, 16
```

### Example 2 (using speed control)

```
EMWriteScreen "list", 19, 7
EMSendKey "@E"
EMWaitReady -1  'use Speed Control to determine script delay
EMWriteScreen "x", 4, 16
```

### Example 3 (non-IBM hosts)

```
EMWrite "list", 19, 7
```

```
EMSendKey "root"
EMWaitReady 0, 100  'wait 100 milliseconds before executing the next command
EMSendKey "@E"
EMWaitReady 0, 100  'wait 100 milliseconds before executing the next command
```

# EMWriteScreen

Pastes the specified text into the host screen.

**Parameters**

**`WriteStr`**
Text to place into host screen.

**`RowVal`**
Row position.

**`ColumnVal`**
Column position.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
`EMWriteScreen` can only paste text into unprotected fields in the host screen.

In a BlueZone VT session, the `WriteStr` parameter is only echoed to the VT client screen. The `WriteStr` parameter is never sent to the host.

**Example**

```
EMConnect "A"
EMWriteScreen "myuserid", 6, 53
EMWriteScreen "mypassword", 7, 53
EMSendKey "@E"
EMWaitReady 10, -1
```

# InputBox

Prompts for user input.

**Parameters**

**`PromptStr`**
A text string that describes to the user the type of input requested.

**`TitleStr`**
Text that appears in the dialog box's title bar. If this parameter is omitted, the title bar remains blank.

**Note**

The `TitleStr` parameter was added in version 4.0C1. Earlier scripts that use the optional `DefaultStr` parameter as the second parameter must modify their script by putting a comma in front of the `DefaultStr` parameter making it the third parameter.

**`DefaultStr`**
A text string to initially display in the input edit box.

**Returns**

A text string containing the user's response or an empty string if the user clicks **Cancel**.

**Remarks**

The `InputBox` dialog displays the prompt string, a field for user input, and the **OK** and **Cancel** buttons. Backslash characters (`\`) embedded in `PromptStr` are treated as CRLFs and can be used to control the width and height of the prompt text.

**Example**

```
ResultStr = InputBox( "Enter run date:", "Run Date", "02/07/2000" )
EMWriteScreen ResultStr, 22, 7
EMSendKey "@E"
EMWaitReady 10, 1
```

## MsgBox

Displays a message to the user.

**Parameters**

**`MessageStr`**

Text to display.

**`FlagsVal`**

Value indicating which icon and buttons to display. See the Remarks section below for a complete description of `FlagsVal`.

**Returns**

The following constants are used with the `MsgBox` function to identify which button a user has selected. These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

| Constant | Value | Description |
|---|---|---|
| VbOK | 1 | OK button was clicked. |
| VbCancel | 2 | Cancel button was clicked. |
| VbAbort | 3 | Abort button was clicked. |
| VbRetry | 4 | Retry button was clicked. |
| VbIgnore | 5 | Ignore button was clicked. |
| VbYes | 6 | Yes button was clicked. |
| VbNo | 7 | No button was clicked. |

**Remarks**

The following constants are used with the `MsgBox` function to identify what buttons and icons appear on a message box and which button is the default. In addition, the modality of the `MsgBox` can be specified. Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

| Constant | Value | Description |
|---|---|---|
| VbOKOnly | 0 | Display OK button only. |
| VbOKCancel | 1 | Display OK and Cancel buttons. |
| VbAbortRetryIgnore | 2 | Display Abort, Retry, and Ignore buttons. |
| VbYesNoCancel | 3 | Display Yes, No, and Cancel buttons. |
| VbYesNo | 4 | Display Yes and No buttons. |
| VbRetryCancel | 5 | Display Retry and Cancel buttons. |
| VbCritical | 16 | Display Critical Message icon. |
| VbQuestion | 32 | Display Warning Query icon. |
| VbExclamation | 48 | Display Warning Message icon. |
| VbInformation | 64 | Display Information Message icon. |
| VbDefaultButton1 | 0 | First button is the default. |
| VbDefaultButton2 | 256 | Second button is the default. |
| VbDefaultButton3 | 512 | Third button in the default. |
| VbDefaultButton4 | 768 | Fourth button is the default. |
| VvbApplicationModal | 0 | Application modal. The user must respond to the message box before continuing work in the current application. |
| VbSystemModal | 4096 | System modal. On Win16 systems, all applications are suspended until the user responds to the message box. On Win32 systems, this constant provides an application modal message box that always remains on top of any other programs you may have running. |

The first group of values (0–5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512, 768) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument buttons, use only one number from each group.

**Example**

```
' displays an information message box having the OK and Cancel buttons
Result = MsgBox( "Press OK to continue or Cancel to quit.", 65 )
```

# OpenSession

Starts a BlueZone Display session.

**Parameters**

**SessionTypeVal**
0 - Mainframe; 1 - iSeries; 2 - VT

**SessionIdentifierVal**
1 for S1; 2 for S2; 3 for S3; ..., 99 for S99

**ConfigFileStr**
Name of the BlueZone configuration file.

**TimeoutVal**
Number of seconds before returning with error.

**WaitPaintsVal**
Number of screen paints before proceeding with script.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The `OpenSession` function waits for the specified number of screen paints to occur in the BlueZone session before continuing with script execution. This number is usually one (1) on mainframe systems and two (2) on AS/400 systems. If the number of seconds specified in the `TimeoutVal` parameter elapses before the number of specified screen paints have occurred in the BlueZone session, then `OpenSession` returns with a non-zero error code. The `CloseSession` function can be used to end a session started with the `OpenSession` function.

**Example**

```
UserId = "John"
Password = "Smith"
ResultCode = OpenSession( 0, 2, "System2.Zmd", 60, 2 )
If ( ResultCode ) Then
   MsgBox "Error connecting to host system!", 0
   End Sub
End If

ResultCode = EMConnect( "A" )
If ( ResultCode ) Then
   MsgBox "Error connecting to session A!", 0
   End Sub
End If
'continue with script execution
CloseSession 0, 2
```

# Pause

Suspends script execution based on the value as noted below.

**Parameters**

**PauseTimeVal**
Value to pause. If the value is less than 19, the pause time is counted in seconds. If the value is greater than 19, the pause time is counted in milliseconds.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

In Example 1 below, a value of 3 causes the script to pause for 3 seconds. In Example 2, a value of 500 causes the script to pause for 500 milliseconds which is equal to half a second.

**Example 1**

```
Pause 3
```

**Example 2**

```
Pause 500
```

# PrintString

The `PrintString` method can be used to print the contents of a buffer to your Windows printer. Formatting, like carriage returns and line feeds, can be added through standard scripting methods. See the example below.

**Parameters**

**`PrintStr`**
Name of string to print.

**`FlagVal`**
Specifies whether or not you want the printer dialog to be displayed. Setting this value to a 1 or true causes the standard Windows printer selection dialog to be displayed. Omitting this value or setting it to 0 or false causes the string or string value to be sent directly to the default printer.

**Returns**

None.

**Remarks**

Can be used with or without the Optional Printer Dialog Flag.

Page numbers (including the word "Page") automatically print at the bottom center of each page.

When the data is sent to the printer, a system beep sounds.

**Example**

In the following example, the script checks to see if it is on the correct screen (by searching for the string "===> ENTER") then it goes to position row 18, column 12, of the screen, reads in the next 75 characters and places them into buffer one. Then it goes to position row 21, column 13, of the screen, reads in the next 75 characters and places them into buffer two. Then, it assigns the value of buffer 1 plus a carriage return and line feed, plus an additional line feed, plus the value of buffer 2 to "page". Then it prints the contents of "page" (shown in red).

```
dim TextFound, Buf1, Buf2, page
Sub Main
EMConnect( "A" )
' This is to make sure you are on the correct screen
TextFound = EMSearch( "===> ENTER", 1, 1 )
If TextFound = 0 then
  MsgBox "You are on the correct screen, you may continue!", 4096
  EMReadScreen Buf1, 75, 18, 012
  EMReadScreen Buf2, 75, 21, 013
End If
Page = Page + Buf1 + VbCrLf + VbLF + Buf2
MsgBox "Make sure your printer is ready and click OK to print!", 4096
PrintString Page
End Sub
Main
```

**Note**

Optionally, you can prompt the user with the standard Windows printer selection dialog by including the optional prompt parameter as shown here:

```
PrintString Page, 1
```

**Additional remarks**

The following application of this script describes how to print a customer account number followed by the customer's address:

1. In `EMSearch`, substitute a unique screen identifier for `"===> ENTER"`. Find a text string on the screen where the customer data is displayed. It is good practice to create scripts in such that way that they can only run on the screen that they were designed for.
2. In the first `EMReadScreen`, substitute the exact cursor position where the **Customer Number** field starts. You may have to decrease the number of characters to read.
3. In the second `EMReadScreen`, substitute the exact cursor position where the **Customer Address** field starts.
4. Test the script by running it. Most addresses consist of three lines. You may have to do a little work to get the formatting of the print job exactly the way you want it.

**Tip**

You can also modify this script to "read in" a large amount of text (data), format the text, and send it to your printer.

# Run

Executes a program.

**Parameters**

**`CommandStr`**

Name of program executable and any command-line arguments to run.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

None.

**Example**

```
Run "C:\BlueZone\Bzmd.Exe /s2 /fsystem2.zmd"
Pause 14

ResultCode = EMConnect "A"
if ( ResultCode = 0 ) Then
' interact with host
End If
 CloseSession 0, 2
Run "Calc"
```

# StopScript

Stops a script from continuing.

**Parameters**

Optional non-zero exit code. See non-zero example below.

**Returns**

None.

**Remarks**

Can be used with or without the optional non-zero exit code.

**Example without the optional exit code**

```
Run "C:\BlueZone\Bzmd.Exe /s2 /fsystem2.zmd"
Pause 14

ResultCode = EMConnect "A"

If ( ResultCode <> 0 ) Then
' stop script from executing
StopScript
End If
```

**Example with a non-zero exit code**

```
BeginDialog Dialog1, 0, 0, 201, 72, "Logoff Before Disconnect"
  ButtonGroup ButtonPressed
    OkButton 36, 45, 50, 15
    CancelButton 116, 45, 50, 15
  Text 15, 10, 175, 25, "Please make sure you are logged out of the VM system
before disconnecting.  Press Cancel to abort disconnect."
EndDialog
nRet = Dialog( Dialog1 )
If ( nRet = 0 ) Then
  StopScript 1 'tells BlueZone to abort any disconnect or close
End If
```

# Str

Returns a string representation of the specified value.

**Parameters**

**ValueVal**
Number to convert.

**Returns**
An array of characters that represent the number.

**Remarks**
The `Str` function can be used to convert variable formats.

**Example**

```
EMReadScreen ResultStr, 2, 7, 15
x = Val(ResultStr) / 2
ResultStr = Str( x )
EMWriteScreen ResultStr, 7, 15
```

## Val

Returns a number representation of the specified string.

**Parameters**

**StringStr**
String of characters to convert.

**Returns**
A number representing the specified string.

**Remarks**
The `Val` function can be used to convert variable formats.

**Example**

```
EMReadScreen ResultStr, 2, 7, 15
x = Val(ResultStr) / 2
ResultStr = Str( x )
EMWriteScreen ResultStr, 7, 15
```

# Script Host sample script

```
'

' Sample Visual Basic script using BlueZone Script Host

'

' This sample script will run a BlueZone session and then

' logon to a AS/400 system and write the Job Log to the disk

' file C:\JOBLOG.TXT.

'

' Subroutine Main

Sub Main

' Run BlueZone AS/400 Display
```

```
' with session id of S1
' no config file(use settings from Registry)
' timeout and return error if no signon screen after 30
' seconds
' continue with script execution after host sends 1 screen
' paint
'
Retval = OpenSession( 1, 1, "", 30, 1 )
If ( Retval ) Then
   MsgBox "Error connecting to host!", 48
   EndSub
End If
' connect to session with hllapi id of "A"
' return error if session not found
'
Retval = EMConnect( "A" )
If ( Retval ) Then
   MsgBox "Error connecting to session A!", 48
   EndSub
End If
' logon to host
' then wait for host to unlock the keyboard
'
' Note: Mainframe sessions will need to enter the number
' of keyboard restores sent from the host system as
' parameter 2 in EMWaitReady ... not needed and is 0
' for AS/400 scripting.
'
EMSendKey "guest3@Tguest3@E"
EMWaitReady 10, 0
' go to Display Job Log screen
EMSendKey "1@E"
EMWaitReady 10, 0
```

```
EMSendKey "1@E"
EMWaitReady 10, 0
EMSendKey "10@E"
EMWaitReady 10, 0
' create disk file C:\JOBLOG.TXT
'
Set fso = CreateObject( "Scripting.FileSystemObject" )
Set f = fso.OpenTextFile( "c:\joblog.txt", 2, True )
' write the job log screens to disk
' read the screens until "Bottom" is found at position
' row 19, column 74
'
MoreText = "More.."
BottomText = "Bottom"
While MoreText <> BottomText
   For i = 1 to 24
      EMReadScreen Buf, 80, i, 1
      f.WriteLine Buf
   Next
   f.WriteLine " "
   EMReadScreen MoreText, 6, 19, 74
Wend
' close the file
'
f.Close
' logoff from host
'
EMSendKey "@E"
EMWaitReady 10, 0
EMSendKey "@3"
EMWaitReady 10, 0
EMSendKey "90@E"
```

```
EMWaitReady 10, 0

' close BlueZone AS/400 Display

' having session id of S1

'

CloseSession 1, 1

' end of Subroutine Main

'

End Sub

' run Subroutine Main

'

Main
```

# Chapter 3: BlueZone Dialog Editor

BlueZone Dialog Editor is a Windows-based program that provides a 100% GUI development tool for the dynamic creation of script dialogs. BlueZone Dialog Editor dynamically generates the necessary script code as the dialog is created with the GUI editor. Then, the script code can be cut and pasted into the main body of a master script. BlueZone Dialog Editor is designed to be used in conjunction with the BlueZone Script Host and Debugger environment.

The major features of BlueZone Dialog Editor are:

- Drag-and-drop
- GUI interface for the insertion of dialog controls
- Support for all script dialog controls:
  - Button
  - Text
  - GroupBox
  - EditBox
  - CheckBox
  - RadioButton
  - ListBox
  - Dropdown
  - ListBox
  - ComboBox
  - Static ComboBox
  - Pictures
- Full layout editing controls for layout positioning of objects
- Adjustable layout alignment grid
- Automatic source code generation
- Tab ordering feature
- Copy and paste feature for inserting source code into scripts

## Using BlueZone Dialog Editor

BlueZone Dialog Editor provides a drag-and-drop interface to create dialogs that can be used in any VBScript or JavaScript when used in conjunction with BlueZone Script Host and Debugger.

The following steps provide an overview of how BlueZone Dialog Editor is used.

1. Drag the dialog controls onto the layout.

   This dynamically generates script code that BlueZone Script Host and Debugger can use.
2. Once the dialog is finished, copy the code from the script area and pastes it into the main script code.

   This eliminates the trial and error method of manually coding the dialog and then running to script to see what it looks like.
3. Once the dialog is in the script, the script can be coded to use the values set in the dialog by the controls.

4. Click **OK** to send the control values to the script.

# BlueZone Dialog Editor controls

Dialogs created using BlueZone Dialog Editor can have one or more of the following standard Windows Controls. Refer to Figure 2: Sample dialog to see what each control looks like.

**Group box**
An outlined box used to label controls with related functions.

**Button**
A button control that sends sets a value when pressed.

**Text**
Static text used to label dialog controls.

**Edit box**
A control that accepts text typed by the user.

**Check box**
A control that uses a check mark to indicate the on or "true" value and blank to indicate off or "false" value.

**Radio button**
A control used in a group of two or more used to force the selection of one item in the group.

**List box**
A control containing a static list of items from which the user chooses.

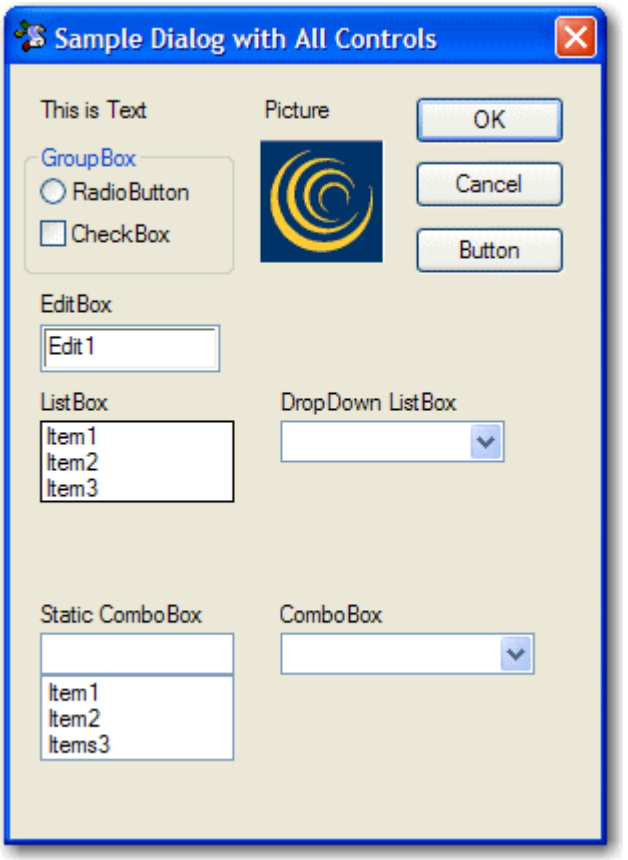
**Drop-down box**
A control containing a list of items that display when the arrow is clicked, "dropping" the box.

**Static combo box**
A control that contains a static list of items from which to choose, but also allows the user to type in a value that is not in the list.

**Combo box**
A drop control that contains a static list of items from which to choose, but also allows the user type in a value that is not in the list. The selection list is displayed when the arrow is clicked, "dropping" the box.

**Picture**
Used to display a bitmap image on the dialog.

Figure 2: Sample dialog




# Starting projects

To start a new dialog project, launch BlueZone Dialog Editor from the BlueZone program group. A blank dialog project opens by default.

To start a new dialog project when an existing project is displayed in Dialog Editor, click **File** ® **New** from the menu bar.

# Saving projects

1. Click **File** ® **Save** or **File** ® **Save As** from the menu bar.
2. When prompted, type a file name. The dialog file is only used to store and recall the dialog project and is not used in conjunction with the final script.

# Opening saved projects

1. Click **File** ® **Open** from the menu bar.
2. Select the desired dialog file (`.dlg`) to open.

BlueZone Dialog Editor also maintains a list of recently accessed projects that can be accessed from **File** ® **Recent Files**.

# Adding dialog titles

By default, the dialog title is `"Dialog 1"`.

To change the dialog title, edit the `BeginDialog` line in the script area of the dialog.

1. Locate the `BeginDialog` line in the script area.
2. Change the default title of `"Dialog 1"` to the desired title. For example, to change the dialog title to `Sample Dialog`, change the following line from:

   ```
   BeginDialog Dialog1, 0, 0, 186, 233, "Dialog 1"
   ```

   To

   ```
   BeginDialog Dialog1, 0, 0, 186, 233, "Sample Dialog"
   ```

# Adding controls to a project

1. Click the desired control on the element tool bar.

   Mousing over each button displays a tool tip label.
2. Click the desired place in the dialog for the selected control.

   The control displays in the dialog. You can resize or move the position of the control.

   The script code for the new control appears automatically in the script area.

# Modifying control attributes

When a control has been placed on the dialog, its attributes can be modified.

1. Right-click or double-click the desired control.

   The control properties window opens.

   The control attributes vary by type, but include the x-y position of the control on the dialog, the variant that accepts the control values at run time, the list of items, and the initial text.
2. Modify the attributes as needed.
3. Click **OK**.

# Testing the dialog

Before copying the dialog text to the script, the dialog can be tested in the Dialog Editor.

Click **Layout** ® **Test Dialog** from the menu bar or click the **Test Dialog** icon  .

The dialog appears allowing you to test the functions of the controls.

# Adding dialogs to scripts

When the dialog is complete, the code is added to the script by selecting all of the text in the script area and pasting it into the main script code in BlueZone Script Host and Debugger. The dialog code can be tested by itself in BlueZone Script Host and Debugger by pasting the text in a new script project, adding the Dialog script command, and clicking **Script** ® **Play Script** from the menu bar or clicking the **Play Script** toolbar icon.

**Note**

The code generated in BlueZone Dialog Editor is only the dialog definition. To invoke the dialog, use the dialog script command shown at the bottom of the following dialog definition. For example, `Dialog Dialog1`.

## Sample dialog text

```
BeginDialog Dialog1, 0, 0, 186, 233, "Sample Dialog with All Controls"
  ButtonGroup ButtonPressed
    OkButton 135, 10, 50, 15
    CancelButton 135, 30, 50, 15
  OptionGroup RadioGroup1
    RadioButton 10, 30, 60, 10, "RadioButton", Radio1
  CheckBox 10, 40, 50, 15, "CheckBox", Check1
  Text 10, 5, 70, 10, "This is Text"
  EditBox 10, 80, 60, 15, Edit1
  ListBox 10, 110, 65, 40, "Item1"+chr(9)+"Item2"+chr(9)+"Item3", List1
  DropListBox 90, 110, 75, 40, "Item1"+chr(9)+"Item2"+chr(9)+"Item3", List2
  StaticComboBox 10, 175, 65, 45, "Item1"+chr(9)+"Item2"+chr(9)+"Items3", Combo1
  ComboBox 90, 175, 85, 50, "Item1"+chr(9)+"Item2"+chr(9)+"Item3", Combo2
  Text 90, 100, 65, 10, "DropDown ListBox"
  Text 90, 165, 80, 10, "ComboBox"
  GroupBox 5, 20, 70, 40, "GroupBox"
  Text 10, 100, 50, 10, "ListBox"
  Text 10, 70, 50, 10, "EditBox"
  Text 10, 165, 65, 10, "Static ComboBox"
  ButtonGroup ButtonPressed
    PushButton 135, 50, 50, 15, "Button", Button3
  Text 90, 5, 30, 10, "Picture"
  Picture 85, 25, 70, 60, "bz1.bmp", Picture2
EndDialog
Dialog Dialog1
```

# Chapter 4: BlueZone Host Automation Object

The BlueZone Host Automation Object is a Component Object Model (COM) software component for 32-bit Windows platforms. BlueZone Host Automation Object can be utilized by any COM container application like Visual Basic, Microsoft Excel, and Microsoft Word to enable communications between PCs running BlueZone Display emulation software products and IBM mainframe and iSeries systems. With BlueZone Host Automation Object, applications can execute common tasks on various host systems, automate user input, obtain data from host systems, initiate file transfers, and more.

The BlueZone Host Automation Object is a language-independent software component. Programs written in Visual Basic, Pascal, C, C++, etc. can invoke the BlueZone Host Automation Object to communicate with the host system. In addition, the BlueZone Host Automation Object can be incorporated into many popular word processing, database and spreadsheet macros, and run by any ActiveX scripting engine, including BlueZone Script Host and Debugger.

The BlueZone Host Automation Object utilizes capabilities of the BlueZone file mapping (shared memory), DDE (Dynamic Data Exchange), and HLLAPI (High-Level Language API) interfaces. In addition to the container's properties and methods, the BlueZone Host Automation Object adds objects, properties, and methods that enable interaction with the BlueZone session and the host system.

In order for the BlueZone Host Automation Object to communicate with a BlueZone display emulation session, you must enable the BlueZone DDE interface.

## Advantages of using BlueZone Host Automation Object

- Easier to implement than HLLAPI or DDE and provides greater functionality.
- Allows easy integration with any COM compliant application.
- Language independent.
- A COM interface is available with the BlueZone Migration Toolkit that allows easy migration to BlueZone. This interface is compatible with the following emulators:
  - Attachmate Extra
  - Attachmate Reflection
  - NetManage Cameleon
  - IBM Personal Communications
  - OHIO (Open Host Interface Objects)

## Disadvantages of using BlueZone Host Automation Object

BlueZone Host Automation Object is a development tool that requires familiarity with programming to implement.

# Configuring BlueZone

In order for the BlueZone Host Automation Object to communicate with a BlueZone Display emulation session, you must enable the BlueZone DDE interface.

1. Launch the desired BlueZone Display session.
2. From the menu bar, click **Options** ® **API**.
   The API Properties window opens.

3. On the **Options** tab, locate the **Enable DDE Server Interface** check box and ensure that the check box is enabled.
4. Locate the **Auto-Assign HLLAPI Names ('A' for S1, 'B' for S2, etc.)** check box and ensure that it is enabled.
5. Click **OK**.
   The BlueZone DDE Server Interface is now enabled.
6. Save the configuration.

# BlueZone Object methods

The BlueZone Host Automation object methods can be incorporated into scripts just like any other scripting command. Select a method to view details about functionality, parameters, and return codes. Examples are written in VBScript.

```
Val CloseSession( SessionTypeVal, SessionIdentifierVal );

Val Connect( SessionShortNameStr,  !, [Blank] );

Val Connected(  );

Copy( ScreenVal );

Val(get) CursorColumn( ColumnVal(set) );

Val(get) CursorRow( RowVal(set) );

Val DeleteSession( SessionNameStr );

Val Disconnect;

Exit(  );

ExtendSelectionRect( RowVal, ColumnVal );

Object Field(  );

Val Focus;

Str GetClipboardText(  );

Val GetCursor( RowVal, ColumnVal );

Str GetFolderName( TitleStr, DescriptionStr, FolderStr );

Str GetOpenFilename( FileFilterStr, FilterIndexVal, TitleStr, ButtonTextStr );

Str GetSaveAsFilename( InitialFilenameStr, FileFilterStr, FilterIndexVal,
TitleStr, ButtonTextStr );

Val GetSessionId(  );

Val GetSessionName(  );

Str InputBox( PromptStr, DefaultStr );

Val LockKeyboard( LockVal );

Val MsgBox( MessageStr, FlagVal );

Str NewSession( SessionTypeVal, ConfigFileStr, LockKeyboardVal );
```

```
Val OpenSession( SessionTypeVal, SessionIdentifierVal, ConfigFileStr,
TimeoutVal, WaitPaintVal );

Paste(  );

Val Pause( PauseTimeVal );

Val PrintScreen( HeaderStr, NumCopiesVal );

Val(get) PSCursorPos( PositionVal(set) );

Str PSGetText( LengthVal, PositionVal );

Val PSSearch( SearchStr, StartPosVal );

Val PSSetText( TextStr, PositionVal );

Val QueryFieldAttribute( PositionVal );

Quit(  );

Val ReadScreen( BufferStr, LengthVal, RowVal, ColumnVal );

Str ReceiveFile( ReceiveStr ); *

Str Run( CommandStr );

RunExternalMacro( ProjectStr, MacroStr, MacroParamStr );

RunMacro( MacroStr, MacroParamStr );

RunScript( ScriptStr );

Val Search( SearchStr, RowVal, ColumnVal );

Str SendFile( SendStr ); *

Str SendKey( KeyStr );

Val SetBrowserWnd;

SetClipboardText( TextStr );

Val SetCursor( RowVal, ColumnVal );

Str SetDLLName( NameStr );

Str SetHostPort( SessionType, SessionId, TCPPort );

SetSelectionStartPos( RowVal, ColumnVal );

Str StartTrace( FileName );

Val Status(  );

Val StopTrace;

Val Str( ValueVal );

Val TCPSetParameters( HostAddrStr, ModelTypeVal, PortVal );

Val(get) TelnetEncryption ( EncryptionState(set) );

Val TypePassword( AccountNameStr );
```

```
Val TypeUserName( AccountNameStr );

Val Val(StringVal );

ViewStatus(  );

Val Wait( WaitVal );

Val WaitCursor( TimeoutVal, RowVal, ColumnVal, ExtraWaitVal );

Str WaitForKeys( TimoutVal, KeysStr );

Val WaitForReady(  );

Val WaitForText( TextStr, RowVal, ColumnVal, TimeOutVal );

Val WaitReady( TimeoutVal, ExtraWaitVal );

Object Window(  );

Val WindowHandle(  );

Val(get) WindowState( StateVal(set) );

Val WriteScreen( WriteStr, RowVal, ColumnVal );
```

* Not supported in BlueZone VT

# CloseSession

Disconnects from the host system and closes the BlueZone Display session window.

**Parameters**

**SessionTypeVal**
0 - Mainframe; 1 - iSeries; 2 - VT

**SessionIdentifierVal**
1 for S1; 2 for S2; 3 for S3; etc.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The `CloseSession` function can be used to end a session started with the `OpenSession` function.
Starting with BlueZone version 3.3, the `CloseSession` method works with BlueZone Web-to-Host when using the Embedded Client Mode.

**Note**

The embedded BlueZone session must first be connected to the program or script by using the Connect method.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.OpenSession 0, 2, "System2.Zmd", 30, 1
ResultCode = Host.Connect( "A" )
If ( ResultCode = 0 ) Then
   ' interact with host
```

```
End If
Host.CloseSession 0, 1
```

# Connect

Opens a conversation with the BlueZone Display session. The `Connect` command must be called before any other BlueZone Host Automation object methods that access data in the host screen.

Connect auto-connects to the BlueZone session that launched the BlueZone Object or it searches for the first available session if launched from BlueZone desktop when no short name session identifier is specified.

**Parameters**

**`SessionShortName`**
Uniquely identifies the BlueZone Display session. The session name corresponds to the **HLLAPI Short Name Session Identifier** configured in the **Options** ® **API** settings in the BlueZone Display emulator. See Example 1 below.

As an option, when the BlueZone session is embedded, using an exclamation point (!) can be used in place of an actual `SessionShortName`. When ! is used, the BZHAO auto-determines the session name. See Example 2 below.

**`[Blank]`**
If the `SessionShortName` parameter is omitted completely, the BZHAO connects to the session that is running in the foreground. If no session is running in the foreground, then the first session found is used.

**`ConnectRetryTimeout`**
Optional: Used to set the time in seconds that `Connect` spends attempting to connect to the BlueZone session. Also, as an option, a zero (0) can be used to cause the `Connect` to abort if the first attempt fails. See Example 3 below.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The script can only be connected to one display session at a time. When connecting to multiple sessions, any previously connected session is automatically disconnected.

**Example 1**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
UserId = "John"
Password = "Smith"
ResultCode = Host.Connect( "A" )
If ( ResultCode = 0 ) Then
   Host.WriteScreen Userid, 5, 63
   Host.WriteScreen Password, 6, 63
   Host.SendKey "@E"
   Host.WaitReady 10, 1
End If
```

**Example 2**

```
Dim host, SessionName, SessionId
Sub LoginToHost
   Set Host = CreateObject( "BZWhll.WhllObj" )
   Host.SetBrowserWnd window.top
   Host.ConnectToHost 2, 0 'zero means BZHAO must auto-determine session id
   Host.Connect "!"        '! means BZHAO must auto-determine session name
   SessionName = Host.GetSessionName()
   SessionId = Host.GetSessionId()
   Host.addConnectionCallback 2, SessionId, "ConnectionCallback"
   'additional login statements
   Host.Disconnect  'break link with session when done
End Sub
Sub ConnectionCallback
   Host.Connect SessionName   'connect to the same session
   'additional logoff statements
   Host.Disconnect
End Sub
```

**Example 3**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
UserId = "John"
Password = "Smith"
ResultCode = Host.Connect( "A", 5 ) 'try to connect for 5 seconds
If ( ResultCode = 0 ) Then
   Host.WriteScreen Userid, 5, 63
   Host.WriteScreen Password, 6, 63
   Host.SendKey "@E"
   Host.WaitReady 10, 1
End If
```

# Connected

Retrieves the session's host connection status.

**Parameters**

None.

**Returns**

True if the session is connected to the host system, or False otherwise.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
If ( bzhao.Connected() ) Then
   bzhao.SendKey "logon user<Enter>"
End If
```

# Copy

Executes the **Edit** ® **Copy to Clipboard** function.

**Parameters**

**ScreenVal**

If set to 32 then the **Edit** ® **Select All** function is executed before the Copy.

**Returns**

None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.Copy 0
```

# CursorColumn

A property to get or set the cursor's column position.

**Parameters**

**`ColumnVal`**
When setting the cursor column, used to specify the new position.

**Returns**
The cursor's column position when using the get property.

**Example (get)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
column = bzhao.CursorColumn
MsgBox "The cursor is on column " & column
```

**Example (set)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.CursorColumn = 27
```

# CursorRow

A property to get or set the cursor's row position.

**Parameters**

**`RowVal`**
When setting the cursor row, used to specify the new position.

**Returns**
The cursor's row position when using the get property.

**Example (get)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
row = bzhao.CursorRow
MsgBox "The cursor is on row " & row
```

**Example (set)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.CursorRow = 4
```

# DeleteSession

Closes an emulation session.

**Parameters**

**`SessionNameStr`**
String containing the short name session identifier of the session to close.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Example**

```
Set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.DeleteSession "A"
```

## Disconnect

The `Disconnect` method is used in conjunction with the `Connect` method, to halt communication between the Host Automation Object and the currently connected BlueZone session. `Disconnect` must be called when a subroutine or module is done with the host session.

To reestablish communication with the same BlueZone Session or a new BlueZone session, use the Connect method. See the Connect method for more information.

**Parameters**

None.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The `Disconnect` method must be used in conjunction with the `Connect` method as shown in the following example.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
UserId = "John"
Password = "Smith"
ResultCode = Host.Connect( "A" )
If ( ResultCode = 0 ) Then
   Host.WriteScreen Userid, 5, 63
   Host.WriteScreen Password, 6, 63
   Host.SendKey "@E"
   Host.WaitReady 10, 1
   Host.Disconnect
End If
```

## Exit

Executes the **File** ® **Exit** function.

**Parameters**

None.

**Returns**

None.

**Remarks**

The `Exit` method can be used to close a BlueZone session.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.Exit
```

# ExtendSelectionRect

Sets the end position of an edit selection.

**Parameters**

**RowVal**
The end row of the edit selection.

**ColumnVal**
The end column of the edit selection.

**Returns**
None.

**Remarks**
If there is no edit selection when this method is called, then the edit selection start position is set to row 1, column 1.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.SetSelectionStartPos 2, 1
bzhao.ExtendSelectionRect 12, 80
```

# Field

Retrieves a handle to the Field automation object.

**Field object methods**
`First`: Sets the field properties of the first field.
`Next`: Sets the field properties to the next field.
`Prev`: Sets the field properties to the previous field.

**Field object properties**
`Length`: Field length.
`Modified`: True if the field has been modified, False otherwise.
`Pos`: Field position.
`Protected`: True if the field is protected, False otherwise.
`Text`: Used to set or retrieve the field text.
`Type`: Field type. 0 - AlphaNumeric, 1 - Alpha, 2 - Numeric Shift, 3 - Numeric All, 5 - Numeric Dup, 7 - Numeric Sign.
`Visible`: True if a display field, or False if a non-display(hidden) field.

**Returns**
A Field object handle.

**Remarks**
The `First()` method must be called initially to set the properties of the first field in the host screen.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
set fld = bzhao.Field()
fld.First
While ( fld.Protected )
   MsgBox fld.Text
   fld.Next
Wend
MsgBox "The first unprotected field is at position " & fld.Pos
```

# Focus

Controls window and keyboard focus.

**Parameters**

None.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

Gives Windows focus and keyboard focus to the host session that you are connected to.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.Focus
```

# GetClipboardText

Copies the contents of the clipboard to a string variable.

**Parameters**

None.

**Returns**

The text retrieved from the clipboard.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
Text1 = bzhao.GetClipboardText()
MsgBox "The text on the Clipboard is: " & Text1
```

# GetCursor

Retrieves the host screen cursor position.

**Parameters**

**`RowVal`**

Variable to contain cursor's row position.

**`ColumnVal`**

Variable to contain cursor's column position.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

When the `GetCursor` function returns, the `RowVal` and `ColumnVal` variables contain the host screen cursor position.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.GetCursor row, col
Host.MsgBox "Cursor is at row " & row & ", column " & col, 0
```

# GetFolderName

Invokes the common Folder dialog.

**Parameters**

**`TitleStr`**
Optional: Title to display in the dialog caption. If omitted, the caption displays "`Browse for Folder`".

**`DescriptionStr`**
Optional: Custom text to display in the dialog window. If omitted, the window displays "`Currently selected folder:`".

**`FolderStr`**
Optional: The default path of the folder to select.

**Returns**
The full path and name of the selected folder, or an empty string if the selection was canceled.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
Folder1 = bzhao.GetFolderName("Base Folder Selection", "Select the
root folder to use:", "c:\")
MsgBox Folder1
```

# GetOpenFilename

Invokes the common Open file dialog.

**Parameters**

**`FileFilterStr`**
Optional: A string of comma-delimited filter pairs used to control which files are initially displayed. For example, "`Text Files,*.txt, Bitmap Files,*.bmp`". If omitted then the "`All Files,*.*`" filter is used.

**`FilterIndexVal`**
Optional: The zero-based index of the filter to use.

**`TitleStr`**
Optional: Title to display in the dialog caption. If omitted, the caption displays "`Open`".

**`ButtonTextStr`**
Optional: Custom text to display in the select button. If omitted, the button displays "`Open`".

**Returns**
The full path and name of the selected file, or an empty string if the selection was canceled.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
File1 = bzhao.GetOpenFileName()
MsgBox File1
```

## GetSaveAsFilename

Invokes the common Save As file dialog.

**Parameters**

**`InitialFilenameStr`**
Optional: The default path and/or file name to select.

**`FileFilterStr`**
Optional: A string of comma-delimited filter pairs used to control which files are initially displayed. For example, `"Text Files,*.txt, Bitmap Files,*.bmp"`. If omitted then the `"All Files,*.*"` filter will be used.

**`FilterIndexVal`**
Optional: The zero-based index of the filter to use.

**`TitleStr`**
Optional: Title to display in the dialog caption. If omitted, the caption displays `"Save As"`.

**`ButtonTextStr`**
Optional: Custom text to display in the select button. If omitted, the button displays `"Save"`.

**Returns**
The full path and name of the selected file, or an empty string if the selection was canceled.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
File1 = bzhao.GetSaveFileName()
MsgBox File1
```

## GetSessionId

Returns the session identifier (1,2,3,etc.) of the currently connected session. This method can only be used after a successful `Connect()` and can be used in subsequent calls that require a session identifier parameter.

**Parameters**
None.

**Remarks**
Although, while not necessary, this method can improve the performance of the script, because the BZHAO does not need to enumerate child windows of the browser when attempting to auto-locate the BlueZone sessions.

**Example**

```
Dim host, SessionName, SessionId
Sub LoginToHost
   Set Host = CreateObject( "BZWhll.WhllObj" )
   Host.SetBrowserWnd window.top
   Host.ConnectToHost 2, 0 'zero means bzhao must auto-determine session id
   Host.Connect "!"        '! means bzhao must auto-determine session name
   SessionName = Host.GetSessionName()
   SessionId = Host.GetSessionId()
   Host.addConnectionCallback 2, SessionId, "ConnectionCallback"
   'additional login statements
   Host.Disconnect  'break link with session when done
End Sub
Sub ConnectionCallback
   Host.Connect SessionName   'connect to the same session
```

```
   'additional logoff statements
   Host.Disconnect
End Sub
```

# GetSessionName

Returns the HLLAPI Short Name ("A","B","C",etc.) of the currently connected BlueZone session. This method can only be used after a successful `Connect()` and can be used in subsequent calls that require a session name parameter.

**Parameters**

None.

**Remarks**

Although, while not necessary, this method can improve the performance of the script, because the BZHAO does not need to enumerate child windows of the browser when attempting to auto-locate the BlueZone sessions.

**Example**

```
Dim host, SessionName, SessionId
Sub LoginToHost
   Set Host = CreateObject( "BZWhll.WhllObj" )
   Host.SetBrowserWnd window.top
   Host.ConnectToHost 2, 0 'zero means bzhao must auto-determine session id
   Host.Connect "!"        '! means bzhao must auto-determine session name
   SessionName = Host.GetSessionName()
   SessionId = Host.GetSessionId()
   Host.addConnectionCallback 2, SessionId, "ConnectionCallback"
   'additional login statements
   Host.Disconnect  'break link with session when done
End Sub
Sub ConnectionCallback
   Host.Connect SessionName   'connect to the same session
   'additional logoff statements
   Host.Disconnect
End Sub
```

# InputBox

Prompts for user input.

**Parameters**

**`PromptStr`**

A text string that describes to the user the type of input requested.

**`DefaultStr`**

A text string to initially display in the input edit box.

**Returns**

A text string containing the user's response, or an empty string if the user clicks the Cancel button.

**Remarks**

The `InputBox` dialog displays the prompt string, an edit box for user input, and the OK and Cancel buttons. Backslash characters (`\`) embedded in `PromptStr` are treated as CRLFs and can be used to control the width and height of the prompt text.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
```

```
ResultStr = Host.InputBox( "Enter run date:", "08/04/2004" )
Host.WriteScreen ResultStr, 22, 7
Host.SendKey "@E"
Host.WaitReady 10, 1
```

# LockKeyboard

When launching a BlueZone session from a web page via an Object Tag, the <PARAM NAME="LockKeyboard" VALUE="Yes"> PARAM name VALUE pair can be manually added to the Object Tag in order to initially set the keyboard to state "locked." When a session's keyboard is locked, all user keyboard input is ignored. This feature is useful when a process is automating a session after it launches, such as through a web page script, or when BlueZone sessions itself is auto-playing a script or macro. Locking the keyboard prohibits user input during session startup and automation.

However, in order for the user to be able to use BlueZone when the process is finished, you must unlock the keyboard To unlock the keyboard, an external process can call the `LockKeyboard( False )` method of the BlueZone Host Automation Object as shown in the example below.

**Note**

If BlueZone is playing a script or macro, BlueZone auto-unlocks the keyboard when the script or macro completes. The `LockKeyboard( False )` method is only needed to unlock the BlueZone keyboard when a process external to BlueZone has completed.

**Parameters**

**`LockVal`**
True or False

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
Setting to True locks the keyboard and setting it to False unlocks the keyboard.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.LockKeyboard( False )
Host.MsgBox( "The keyboard has been unlocked! BlueZone is ready for your input.", 0 )
```

# MsgBox

Displays a message to the user.

**Parameters**

**`MessageStr`**
Text to display.

**`FlagsVal`**
Value indicating which icon and buttons to display. See the Remarks section below for a complete description of `FlagsVal`.

**`Title`**
Optional parameter. Designate the dialog box title. If you omit the argument title, `MsgBox` has the default title "BZWhll:MsgBox".

**Returns**

The following constants are used with the `MsgBox` function to identify which button a user has selected. These constants are only available when your project has an explicit reference to the appropriate type library containing these constant definitions. For VBScript, you must explicitly declare these constants in your code.

| Constant | Value | Description |
|---|---|---|
| VbOK | 1 | OK button was clicked. |
| VbCancel | 2 | Cancel button was clicked. |
| VbAbort | 3 | Abort button was clicked. |
| VbRetry | 4 | Retry button was clicked. |
| VbIgnore | 5 | Ignore button was clicked. |
| VbYes | 6 | Yes button was clicked. |
| VbNo | 7 | No button was clicked. |

**Remarks**

The following constants are used with the MsgBox function to identify what buttons and icons appear on a message box and which button is the default. In addition, the modality of the MsgBox can be specified. Since these constants are built into VBScript, you don't have to define them before using them. Use them anywhere in your code to represent the values shown for each.

| Constant | Value | Description |
|---|---|---|
| VbOKOnly | 0 | Display OK button only. |
| VbOKCancel | 1 | Display OK and Cancel buttons. |
| VbAbortRetryIgnore | 2 | Display Abort, Retry, and Ignore buttons. |
| VbYesNoCancel | 3 | Display Yes, No, and Cancel buttons. |
| VbYesNo | 4 | Display Yes and No buttons. |
| VbRetryCancel | 5 | Display Retry and Cancel buttons. |
| VbCritical | 16 | Display Critical Message icon. |
| VbQuestion | 32 | Display Warning Query icon. |
| VbExclamation | 48 | Display Warning Message icon. |
| VbInformation | 64 | Display Information Message icon. |
| VbDefaultButton1 | 0 | First button is the default. |
| VbDefaultButton2 | 256 | Second button is the default. |
| VbDefaultButton3 | 512 | Third button is the default. |
| VbDefaultButton4 | 768 | Fourth button is the default. |
| VvbApplicationModal | 0 | Application modal. The user must respond to the message box before continuing work in the current application. |
| VbSystemModal | 4096 | System modal. On Win16 systems, all applications are suspended until the user responds to the message box. On Win32 systems, this constant provides an application modal message box that always remains on top of any other programs you may have running. |

The first group of values (0–5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512, 768) determines which button is the default; and the fourth group (0, 4096) determines the modality of the message box. When adding numbers to create a final value for the argument buttons, use only one number from each group.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
 'displays an information message box having the OK and Cancel buttons
Result = Host.MsgBox( "Press OK to continue or
Cancel to quit.", 65 )
```

# NewSession

Launches a new emulation session.

**Parameters**

**SessionTypeVal**
1 – Mainframe; 2 – iSeries; 3 – VT

**ConfigFileStr**
Optional: Name of the profile containing the session settings.

**LockKeyboardVal**
Optional: If True, locks the user's keyboard until a call to `LockKeyboard( False )` is made.

**Returns**
String containing the session's short name session identifier to be used with the `Connect` and/or `DeleteSession` methods.

**Example**

```
Set bzhao = CreateObject("BZWhll.WhllObj")
SessName = bzhao.NewSession( 1, "profile.zmd" )
bzhao.Connect SessName
bzhao.SendKey "logon user<Enter>"
```

# OpenSession

Starts a BlueZone Display session.

**Parameters**

**SessionTypeVal**
0 - Mainframe; 1 - iSeries; 2 - VT

**SessionIdentifierVal**
1 for S1: 2 for S2; 3 for S3; etc.

**ConfigFileStr**
Name of the BlueZone Configuration File.

**TimeoutVal**
Number of seconds before returning with error.

**WaitPaintsVal**
Number of screen paints before proceeding with script.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The `OpenSession` function waits for the specified number of screen paints to occur in the BlueZone session before continuing with script execution. This number is usually one (1) on mainframe systems and two (2) on iSeries systems. If the number of seconds specified in the `TimeoutVal` parameter elapses before the number of specified screen paints have occurred in the BlueZone session, then `OpenSession` returns with a non-zero error code.
The `CloseSession` function can be used to end a session started with the `OpenSession` function.

**Example**

```
UserId = "John"
Password = "Smith"
Set Host = CreateObject( "BZWhll.WhllObj" )
ResultCode = Host.OpenSession( 0, 2, "System2.zmd", 60, 2 )
If ( ResultCode ) Then
```

```
   Host.MsgBox "Error connecting to host system!", 0
   End Sub
End If
ResultCode = Host.Connect( "A" )
If ( ResultCode ) Then
   Host.MsgBox "Error connecting to session A!", 0
   End Sub
End If
 'continue with script execution
Host.CloseSession 0, 2
```

# Paste

Executes the **Edit** ® **Paste** function.

**Parameters**

None.

**Returns**

None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.SetCursor 7, 45
bzhao.Paste
```

# Pause

Suspends script execution based on the value as noted below.

**Parameters**

**PauseTimeVal**

Value to pause. If the value is less than 19, the pause time is counted in seconds. If the value is greater than 19, the pause time is counted in milliseconds.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

For example, in Example 1 below, a value of 3 causes the script to pause for 3 seconds. In Example 2, a value of 500 causes the script to pause for 500 milliseconds which is equal to half a second.

**Example 1**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Pause 3
```

**Example 2**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Pause 500
```

# PrintScreen

Executes the **File** ® **Print Screen** function.

**Parameters**

**HeaderStr**
Optional: Page header.

**NumCopiesVal**
Optional: Number of copies to print.

**Returns**
0 on success, or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.PrintScreen
```

# PSCursorPos

A property to get or set the cursor position.

**Parameters**

**PositionVal**
When setting the cursor position, used to specify the new position.

**Returns**
The current cursor position when using the get property.

**Remarks**
The cursor position starts at 1 in the upper-left corner of the window (row 1, column 1), and ends at the bottom-right of the window (max row times max column). For example, for a Model 2 - 24 x 80 screen, the last position is 1920.

**Example (get)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
pos = bzhao.PSCursorPos
MsgBox "The cursor position is " & pos
```

**Example (set)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.PSCursorPos = 1841
```

# PSGetText

Reads text from the host screen into a variable.

**Parameters**

**LengthVal**
The number of characters to read.

**PositionVal**
The position in the host screen to start reading.

**Returns**
String containing the text.

**Remarks**

The screen position starts at 1 in the upper-left corner of the window (row 1, column 1), and ends at the bottom-right of the window (max row times max column). For example, for a Model 2 - 24 x 80 screen, the last position is 1920.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
TextVar = bzhao.PSGetText( 5, 34 )
MsgBox TextVar
```

## PSSearch

Performs a case-sensitive search for an occurrence of text in the host screen.

**Parameters**

**SearchStr**
The text to search for.

**StartPosVal**
The position in the host screen to begin the search.

**Returns**

The position in the host screen where the text was found, or 0 if not found.

**Remarks**

The screen position starts at 1 in the upper-left corner of the window (row 1, column 1), and ends at the bottom-right of the window (max row times max column). For example, for a Model 2 - 24 x 80 screen, the last position is 1920.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
result = PSSearch( "Logon", 1 )
If ( result ) Then
   MsgBox "Logon found at position " & result
Else
   MsgBox "Logon not found"
End If
```

## PSSetText

Writes text to an unprotected area of the host screen.

**Parameters**

**TextStr**
String containing the text to write.

**PositionVal**
The position in the host screen to start writing.

**Returns**

0 on Success, or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The screen position starts at 1 in the upper-left corner of the window (row 1, column 1), and ends at the bottom-right of the window (max row times max column). For example, for a Model 2 - 24 x 80 screen, the last position is 1920.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.PSSetText( "UserId", bzhao.PSCursorPos )
```

# QueryFieldAttribute

Returns the host field attribute value associated with the specified screen position.

**Parameters**

**`PositionVal`**

The zero-based screen position to query.

**Returns**

The host field attribute value in big-endian format:

Table 2: 3270 Field Attributes

| Bit position | Meaning |
|---|---|
| 0-1 | Both set to 1 (field attribute byte) |
| 2 | Unprotected/protected:<br>0 = Unprotected data field<br>1 = Protected data field |
| 3 | Alpha/numeric:<br>0 = Alphanumeric data<br>1 = Numeric data only |
| 4-5 | I/SPD:<br>00 = Normal intensity, pen not detectable<br>01 = Normal intensity, pen detectable<br>10 = High intensity, pen detectable<br>11 = Non-display, pen not detectable |
| 6 | Reserved |
| 7 | MDT (Modified Data Tag):<br>0 = Field has not been modified<br>1 = Field has been modified |

Table 3: 5250 Field Attributes

| Bit position | Meaning |
|---|---|
| 0 | Field attribute flag:<br>0 = Not a field attribute<br>1 = Field attribute byte |
| 1 | Visibility:<br>0 = Non-display<br>1 = Display |
| 2 | Unprotected/protected:<br>0 = Unprotected data field<br>1 = Protected data field |
| 3 | Intensity:<br>0 = Normal<br>1 = High |
| 4-6 | Field Type:<br>000 = Alphanumeric: all characters allowed<br>001 = Alphabetic only<br>010 = Numeric shift: automatic shift for numerics<br>011 = Numeric only<br>100 = Reserved<br>101 = Digits:<br>110 = Magnetic stripe reader data only<br>111 = Signed Numeric |
| 7 | MDT:<br>0 = Field has not been modified<br>1 = Field has been modified |

**Remarks**

The screen position starts at 0 in the upper-left corner of the window (row 1, column 1), and ends at the bottom-right of the window (max row times max column minus one). For example, for a Model 2 - 24 x 80 screen, the last position is 1919.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
i = 0
While i < 1920
  x = bzhao.QueryFieldAttribute( i )
  if x <> y then
    y = x
    MsgBox "Attribute " & x & " at position " & i
  end if
  i = i + 1
Wend
```

# Quit

Closes all running BlueZone sessions.

**Parameters**

None.

**Returns**

None.

**Example**

```
Example:
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Quit
```

# ReadScreen

Retrieves data from the host screen.

**Parameters**

**`BufferStr`**
Variable to contain host screen data.

**`LengthVal`**
Number of characters to read.

**`RowVal`**
Row position.

**`ColumnVal`**
Column position.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

When the `ReadScreen` function returns, the `BufferStr` variable contains the host screen data.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.ReadScreen Buf, 10, 7, 16
Host.MsgBox "Buf is " & Buf, 0
```

## ReceiveFile

Initiates a IND$FILE file transfer download from the mainframe host system.

**Parameters**

**ReceiveStr**
IND$FILE `Receive` command string.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
Script execution is suspended until the file transfer is complete. The `ReceiveStr` command parameters correspond to what you would normally enter at the DOS prompt. The following example, receives the file `SLSREPRT` from the host system to the PC file `C:\BlueZone\Transfer\Sales.Rpt` through a BlueZone session having the short name session ID of `"A"`:

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.ReceiveFile "C:\BlueZone\Transfer\Sales.Rpt A:SLSREPRT (ASCII CRLF)"
```

## Run

Executes a program.

**Parameters**

**CommandStr**
Name of program executable and any command line arguments to run.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Run "C:\BlueZone\Bzmd.Exe /s2 /fsystem2.zmd"
Host.Pause 7
ResultCode = Host.Connect "A"
If ( ResultCode = 0 ) Then
  'interact with host
End If
Host.CloseSession 0, 2
Host.Run "Calc"
```

## RunExternalMacro

Causes the session to load an external VBA project file, run the specified macro procedure, and then close the VBA project.

**Parameters**

**`ProjectStr`**
Name of the BlueZone VBA project (.bvp) to load.

**`MacroStr`**
The name of the VBA macro to run. The macro procedure must exist in the project file specified by `ProjectStr`.

**`MacroParamStr`**
Optional: Macro parameter to pass to the macro procedure when calling.

**Returns**
None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.RunExternalMacro "Project.bvp", "SubOne"
```

# RunMacro

Runs a VBA macro.

**Parameters**

**`MacroStr`**
The name of the VBA macro to run. The macro procedure must exist in the BlueZone VBA project file loaded by the session.

**`MacroParamStr`**
Optional: Macro parameter to pass to the macro procedure when calling.

**Returns**
None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.RunMacro "SubOne"
```

# RunScript

Runs BlueZone Script Host in quiet-mode to play a script.

**Parameters**

**`ScriptStr`**
The name of the script file to play. If the file exists in the `Scripts` directory, then a path to the script file does not need to be included in the file name.

**Returns**
None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.RunScript "test.vbs"
```

## Search

Searches the host screen for some specified text.

**Parameters**

**SearchStr**
Specifies the text to search for.

**RowVal**
On input, the variable containing the row position where the search is to begin. On output, this variable contains the row position where the text was found.

**ColumnVal**
On input, the variable containing the column position where the search is to begin. On output, this variable contains the column position where the text was found.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
When the `Search` function returns, the `RowVal` and `ColumnVal` variables contain the position where the text was found. If the text was not found, or if the function returned an error, the `RowVal` and `ColumnVal` variables contain zero.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
row = 1
col = 1
Host.Search ">>", row, col
Host.MsgBox ">> is at position " & row & ", " & col, 0
```

## SendFile

Initiates a IND$FILE file transfer upload to the mainframe host system.

**Parameters**

**SendStr**
IND$FILE `Send` command string.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
Script execution is suspended until the file transfer is complete. The `SendStr` command parameters correspond to what you normally enter at the DOS prompt.

The following example, sends the file `SALES.RPT` from your machine to the CMS file `SLSREPRT` on the host system through a BlueZone session having the short name session ID of "`A`":

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.SendFile "C:\BlueZone\Transfer\Sales.Rpt A:SLSREPRT (ASCII CRLF)"
```

# SendKey

Sends a sequence of keys to the display session.

**Parameters**

**KeyStr**
String of key codes. See the Remarks section for a complete listing of valid key codes and descriptions.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The `SendKey` function affects the host screen as if the user were typing on the keyboard.

Refer to the following tables for a list of host function keys and their corresponding codes. If a character is used in the code, then the case of the character is important.

- IBM 3270/5250 send key table, on page 251
- BlueZone VT send key table, on page 253
- UTS send key table, on page 257

**Note**

If you want to use the "at" sign (@) in the data string you must use the two-byte code "@@".

**Example 1**
Example 1 utilizes standard mnemonic syntax. For example, @T for TAB and @E for ENTER.

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.SendKey "John@TSmith@E"
Host.WaitReady 10, 1
```

**Example 2**
Example 2 utilizes friendly syntax. For example, <TAB> for TAB and <Enter> for ENTER.

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.SendKey "John<Tab>Smith<Enter>"
Host.WaitReady 10, 1
```

**Note**

If you are converting macros or scripts to BlueZone, and your existing macros or scripts contain the friendly syntax, there is no need to convert them to mnemonic syntax.

# SetBrowserWnd

Associates an instance of the BZHAO with a browser window that has a BlueZone display session embedded. When using the Sx Object Tag parameter with embedded sessions, the `SetBrowserWnd()` method must be used after each `CreateObject()` call when running multiple browser instances and specifying "!" with `Connect()` and `zero(0)` with `ConnectToHost()`, `addConnectionCallback()`, etc. in the web page scripts.

**Parameters**

**window.top**

**Remarks**

In the following example, the BZHAO uses the `window.top` object to find the BlueZone session that is embedded in the same browser window that is running the script. Note that `window.top` is the only valid parameter for this method.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.SetBrowserWnd window.top
Host.ConnectToHost 2, 0
Host.Connect "!"
Host.ReadScreen Buf, 1920, 1, 1
Host.Disconnect
Host.DisconnectFromHost 2, 0
```

## SetClipboardText

Sets the text contents of the clipboard from a string.

**Parameters**

**`TextStr`**

**Returns**

None.

**Remarks**

To clear the contents of the clipboard, use this method while passing an empty string.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
Text1 = "New Clipboard Text"
bzhao.SetClipboardText Text1
```

## SetCursor

Sets the host screen cursor position.

**Parameters**

**`RowVal`**

Row position.

**`ColumnVal`**

Column position.

**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

BlueZone VT attempts to move the cursor on the screen by sending cursor movement commands to the host. Not all VT applications/screens support cursor movement commands.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.SetCursor 6, 53
Host.SendKey "myuserid@Tmypassword@E"
Host.WaitReady 10, 1
```

# SetDLLName

Controls the inter-process communications mechanism between the BlueZone Host Automation Object and the BlueZone display session.

**Parameters**

**NameStr**
Name of the BlueZone API Dynamic Link Library for inter-process communications.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
By default the BHAO uses the BlueZone `Whllapi.Dll` module for inter-process communications. The BlueZone `Whllapi.Dll` uses the Windows `Ddeml.Dll` to initiate data transactions to the BlueZone session. Some applications can suspend thread execution and inadvertently inhibit the Windows `Ddeml.Dll` from performing data transactions. The `SetDLLName` function can be used to load the BlueZone `Whlapi32.Dll` module which uses file mapping (shared memory) for inter-process communications to the BlueZone display session.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.SetDLLName "Whlapi32"
```

# SetHostPort

Allows the setting of the host TCP port. You must also set the BlueZone session type and session ID.

**Parameters**

**SessionType**
0 - Mainframe; 1 - iSeries; 2 - VT

**SessionId**
1 for S1; 2 for S2; 3 for S3, etc.

**TCPPort**
TCP Port Number

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.SetHostPort 0, 1, 2099
```

# SetSelectionStartPos

Sets the start position of an edit selection.

**Parameters**

**RowVal**
The start row of the edit selection.

**ColumnVal**
The start column of the edit selection.

**Returns**
None.

**Remarks**
A one character edit selection is created. The edit selection can be extended by calling the `ExtendSelectionRect` method.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.SetSelectionStartPos 2, 1
bzhao.ExtendSelectionRect 2, 9
```

## StartTrace

Starts a BlueZone trace. This command must be used in conjunction with `StopTrace`.

**Parameters**

**FileName**
Name that you want to use for the trace file. The trace file is written to the BlueZone `Scripts` folder not the `Traces` folder. Also, if you prefer, you can specify the full path location for the trace file.

**Returns**
None.

**Remarks**
Used for troubleshooting purposes. It is best to place `StartScript` before you make the connection to the host so that you capture all important events from the beginning.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.StartTrace( "trace.txt" )
Retval = Host.Connect "A"
If ( Retval ) Then
   Host.MsgBox "Error connecting to session A!", 48
   EndSub
End If
```

## Status

Returns the status of the host session.

**Parameters**
None.

**Returns**
0 - Ready
4 - Presentation Space is busy
5 - Keyboard is locked

**Remarks**

While the `Status` method can return the current session status, the `WaitReady` method is the preferred way of waiting for a Ready session status after sending an AID-key to the host.

## StopTrace

Stops a BlueZone trace. Used in conjunction with `StartTrace`.

**Parameters**

None.

**Returns**

None.

**Remarks**

Used for troubleshooting purposes. Place `StopTrace` after you close the BlueZone session so that you capture as much information as possible.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.StartTrace( "tracefile.txt")
Host.connect( "A" )

Place additional script commands here.
Host.CloseSession 1, 1
Host.StopTrace
```

## Str

Returns a string representation of the specified value.

**Parameters**

**`ValueVal`**

Number to convert.

**Returns**

An array of characters that represent the number.

**Remarks**

The `Str` function can be used to convert variable formats.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.ReadScreen ResultStr, 2, 7, 15
x = Val(ResultStr) / 2
ResultStr = Host.Str( x )
Host.WriteScreen ResultStr, 7, 15
```

## TCPSetParameters

Used before calling `NewSession` to set or override connection parameters.

**Parameters**

**HostAddrStr**
Host address string used when establishing a connection to the host system.

**ModelTypeVal**
Optional: Model type value used when setting the default number of rows and columns for the device to emulate:

2 - 24x80
3 - 32x80
4 - 43x80
5 - 27x132

**PortVal**
Optional: Port number used when establishing a connection to the host system.

**Returns**
0 on Success, or a non-zero error code. See Error codes, on page 250 for a complete listing of error code descriptions.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.TCPSetParameters "host.address", 5, 23
SessName = bzhao.NewSession( 1, "default.zmd" )
bzhao.Connect SessName
bzhao.SendKey "logon user<Enter>"
```

## TelnetEncryption

A property to get or set the Telnet encryption type.

**Parameters**
None.

**Returns**
Returns the Telnet encryption type when using the `RetVal` property.

The supported values for Mainframe/iSeries are:

- `0` = Off
- `1` = Implicit SSL/TLS
- `2` = Explicit SSL/TLS

The supported values for VT are:

- `0` = Off
- `1` = SSL
- `2` = TLS
- `3` = SSH

**Examples**
The following example gets the Telnet encryption type:

```
RetVal = bzhao.TelnetEncryption
```

The following example sets the Mainframe encryption type to Explicit SSL/TLS:

```
Bzhao.TelnetEncryption 2
```

# TypePassword

Used to auto-type the password associated with a PasswordVault account name.

**Parameter**

**AccountNameStr**
Optional.

**Returns**
0 if the password was typed successfully, 2 if the prompt was canceled, or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
If the PasswordVault account name parameter is omitted, then BlueZone uses the account name associated with the current host screen. If PasswordVault is not enabled, or if the account name is not found, then this method prompts the user to enter a password to type.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.TypeUserName
bzhao.SendKey "<Tab>"
bzhao.TypePassword
bzhao.SendKey "<Enter>"
```

# TypeUserName

Auto-types the user name associated with a PasswordVault account name.

**Parameters**

**AccountNameStr**
Optional.

**Returns**
0 if the user name was typed successfully, 2 if the prompt was canceled, or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
If the PasswordVault account name parameter is omitted, then BlueZone uses the account name associated with the current host screen. If PasswordVault is not enabled, or if the account name is not found, then this method prompts the user to enter a user name to type.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.TypeUserName
bzhao.SendKey "<Tab>"
bzhao.TypePassword
bzhao.SendKey "<Enter>"
```

# Val

Returns a number representation of the specified string.

**Parameters**

**StringStr**
String of characters to convert.

**Returns**
A number representing the specified string.

**Remarks**
The `Val` function can be used to convert variable formats.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.ReadScreen ResultStr, 2, 7, 15
x = Val(ResultStr) / 2
ResultStr = Host.Str( x )
Host.WriteScreen ResultStr, 7, 15
```

## ViewStatus

Launches the BlueZone Session Manager application.

**Parameters**
None.

**Returns**
None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.ViewStatus
```

## Wait

Suspends script execution for the time specified.

**Parameters**

**`WaitVal`**
The number of seconds to wait before continuing with the script.

## WaitCursor

Suspends script execution until the host screen is ready for keyboard input and the cursor is at the specified location.

**Parameters**

**`TimeoutValue`**
The number of seconds to wait before returning with a `session is busy` error code.

**`RowValue`**
Specifies the cursor row position in the host screen.

**`ColumnValue`**
Specifies the cursor column position is the host screen.

**`ExtraWaitVal`**
The number of milliseconds to validate for a keyboard unlocked status.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The `WaitCursor` function can be used to verify that a specific host screen is being displayed before continuing with script execution.

If `ExtraWaitVal` is a value in the range of 1, 2, 3, … ,10 then script execution is suspended until the host machine sends the specified number of screen writes containing the keyboard restore/unlock command.

If `ExtraWaitVal` is greater than 10, script execution is suspended until the specified number of milliseconds have transpired.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.WriteScreen "list", 19, 7
Host.SendKey "@E"
ResultCode = Host.WaitCursor( 10, 4, 16, 3 )   'wait for 3 keyboard restores
If ( ResultCode != 0 ) Then
   Host.MsgBox "Timeout Error in Script!", 0
End If
Host.SendKey x@E
```

# WaitForKeys

Waits for the user to press keys while the focus is in the session window.

**Parameters**

**`TimeoutVal`**
Optional: The number of seconds to wait. If omitted, the default is to wait forever. The maximum finite wait time is 65535 seconds, approximately 18 hours.

**`KeyStr`**
Optional: A string of characters representing the keys to wait for. If omitted, the default is any key.

**Returns**

An empty string if `KeyStr` was omitted, or if the timeout occurred, or the keys pressed if a match was found.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.Focus
result = bzhao.WaitForKeys( 10000, "logon<Enter>"
If ( result <> "" ) Then
   MsgBox "User typed " & result
Else
   MsgBox "Ten seconds timeout occurred." )
End If
```

# WaitForReady

Used after sending an AID key to wait for the host keyboard to unlock.

**Parameters**

None.

**Returns**

0 on Success, or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect SessName
bzhao.SendKey "logon user<Enter>"
bzhao.WaitForReady
```

## WaitForText

Suspends script execution until the desired text is found in the host screen.

**Parameters**

**TextStr**
The text string that you want to search for in the host screen.

**RowValue**
Specifies the start row position in the host screen where the search is to begin.

**ColumnValue**
Specifies the start column position in the host screen where the search is to begin.

**TimeoutValue**
The number of seconds to wait before returning with a `session is busy` error code.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The `WaitForText` function can be used to verify that a specific host screen is being displayed before continuing with script execution.

**Example**

```
Dim Host
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
ResultCode = Host.WaitForText( "S103XWRM", 1, 1, 5 )  'wait
for text for 5 seconds
If ( ResultCode <> 0 ) Then
   Host.MsgBox "Host is not responding or incorrect Host!", 0
End If
Host.Disconnect
```

## WaitReady

Suspends script execution until the host screen is ready for keyboard input.

**Parameters**

**TimeoutVal**
The number of seconds to wait before returning with a `session is busy` error code.

**ExtraWaitVal**
The number of milliseconds to validate for a keyboard unlocked status. For a detailed explanation, see the Remarks section below.

**Note**

The above features behave differently when scripting non-IBM hosts. See Non-IBM Remarks below.

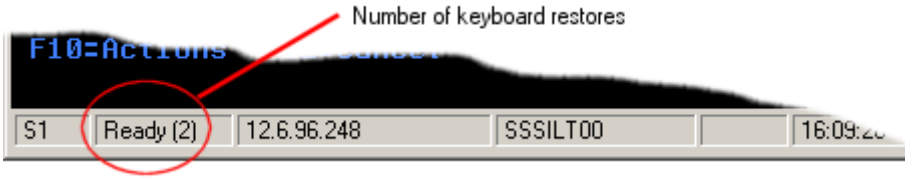
**Returns**

0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**

The `WaitReady` function must be called each time after sending a attention identifier key (such as a PF key) to the display session.

If `ExtraWaitVal` is a value in the range of 1, 2, 3, on up to 50, then script execution is suspended until the host machine sends the specified number of keyboard restores. Refer to the BlueZone status bar to determine the keyboard restore count for a given screen. In the following screen shot **Ready (2)** on the status bar means two keyboard restores were detected when this particular screen was written by the host.




If `ExtraWaitVal` is set to 51 or higher, the operation of the parameter changes to specify the number of milliseconds to wait after the keyboard lock has been detected prior to executing the next script command.

**Note**

`WaitReady` only works after an AID key is sent to the host. If `WaitReady` is used after data is put in a field, but not sent to the host, then the wait count is never reached and the command times out (first parameter). When converting macros and replacing `WaitHostQuiet` with `WaitReady`, ensure the preceding command is Enter, PFKey, Attn, SysReq, or some other AID key that causes the host to write to the screen.

**Non-IBM remark**

`TimeoutVal` and `ExtraWaitVal` behave differently when scripting non-IBM hosts. That is because keyboard locked status is not supported on non-IBM hosts. When scripting on non-IBM hosts, set `TimeoutVal` to `0` and treat `ExtraWaitVal` as a pause before the scripts moves on to the next command. See Example 2 below.

**Example 1 (IBM hosts)**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.WriteScreen "list", 19, 7
Host.SendKey "@E"
Host.WaitReady 10, 2000   'wait 2 seconds for next screen to come down
Host.WriteScreen "x", 4, 16
```

**Example 2 (non-IBM hosts)**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
sub logOn
   Host.SendKey "root"
   Host.WaitReady 0, 100  'wait 100 milliseconds before executing
   the next command
   Host.SendKey "@E"
   Host.WaitReady 0, 100  'wait 100 milliseconds before executing
   the next command
   Host.focus
End Sub
```

# Window

Retrieves a handle to the `Window` automation object.

**Window object properties**

**`Visible`**
True if the window is visible. False otherwise.

**`hWnd`**
The Win32 handle of the window.

**`Active`**
True if the window has keyboard focus. False otherwise.

**`Caption`**
Used to set or retrieve the window's title bar text.

**`Height`**
Window height in pixels.

**`Left`**
Window left position.

**`State`**
0 - normal
1 - minimized
2 - maximized

**`Top`**
Window top position.

**`Width`**
Window width in pixels.

**`ZOrderTop`**
True if the window is at the top of the Z-order. False otherwise.

**Returns**
A window object handle.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
set wnd = bzhao.Window()
wnd.Visible = False     ' hide the session window
wnd.State = 2           ' maximize the window
wnd.Visible = True      ' show the session window
```

# WindowHandle

Returns the window handle of the connected session.

**Parameters**
None.

**Returns**
None.

**Example**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
hwnd = bzhao.WindowHandle()
```

# WindowState

A property to get or set the session's window state.

**Parameters**

**StateVal**
Used when setting the window state:
0 - normal
1 - minimize
2 - maximize

**Returns**
The window's state when using the get property.

**Example (get)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
state = bzhao.WindowState()
MsgBox "The window state is " & state
```

**Example (set)**

```
set bzhao = CreateObject("BZWhll.WhllObj")
bzhao.Connect
bzhao.WindowState = 2
```

# WriteScreen

Pastes specified text in the host screen.

**Parameters**

**WriteStr**
Text to place in host screen.

**RowVal**
Row position.

**ColumnVal**
Column position.

**Returns**
0 for success; or a non-zero error code. Refer to Error codes, on page 250 for a complete listing of error code descriptions.

**Remarks**
The `WriteScreen` method can only paste text in unprotected fields in the host screen.

In a BlueZone VT session, the `WriteStr` parameter is only echoed to the VT client screen. The `WriteStr` parameter is never sent to the host.

**Example**

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Host.WriteScreen "myuserid", 6, 53
Host.WriteScreen "mypassword", 7, 53
Host.SendKey "@E"
Host.WaitReady 10, -1
```

# Host Automation Object samples

## Sample program

In order for the BlueZone Host Automation Object to communicate with a BlueZone display emulation session, you must enable the BlueZone DDE interface.

Refer to Configuring BlueZone, on page 47 for more information.

The following example shows how to incorporate the BlueZone Host Automation Object into a custom C/C++ application:

```
// add BZWhll_i.c to the project as a source file
#include "BZWhll.h"
IWhllObj *pHost = NULL; // pointer to BHAO
// before using BHAO make sure COM is available and initialized
HRESULT hr = CoInitialize( NULL );
if ( FAILED(hr) )
     return 0;
// create an instance of the BHAO object
hr = CoCreateInstance(
               CLSID_WhllObj,
               NULL,
               CLSCTX_INPROC_SERVER,
               IID_IWhllObj,
               (void **)&pHost );
if ( SUCCEEDED(hr) )
{
     // pHost can now be used to access the host system.
     // For purposes of this example, we will just login and logoff
     // of a zSeries system.
     int nRetval; // method return code
     char Buf[ 2048 ]; // 2K buffer
     // connect to the zSeries via BlueZone Mainframe Display
     pHost->Connect( "A", &nRetval );
     if ( nRetval == S_OK )
     {
          // login to the zSeries
          pHost->SendKey( L"logon guest@E", &nRetval );
          pHost->WaitReady( 10, 1, &nRetval );
          pHost->SendKey( L"password@E", &nRetval );
          pHost->WaitReady( 10, 1, &nRetval );
          pHost->SendKey( L"@E", &nRetval );
          pHost->WaitReady( 10, 1, &nRetval );
          // copy the host screen and display in a message box
          pHost->ReadScreen( Buf, 1920, 1, 1, &nRetval );
          MessageBox( NULL, Buf, "BHAO Sample Program", MB_OK );
          // logoff of zSeries
          pHost->SendKey( L"@3", &nRetval );
          pHost->WaitReady( 10, 1, &nRetval );
          pHost->SendKey( L"logoff@E", &nRetval );
          pHost->WaitReady( 10, 1, &nRetval );
     }
     else
     {
          char ErrorMsg[ 64 ];
          wsprintf( ErrorMsg, "Error %d connecting to host system.", nRetval );
          MessageBox( NULL, ErrorMsg, "Connect Error:", MB_OK );
     }
     // when done release the BlueZone Host Automation object
     pHost->Release( );
}
// don't forget to uninitialize COM when you are done
CoUninitialize( );
```

# Sample VBScript

This script is designed to be used with BlueZone Desktop. The script launches an already existing BlueZone iSeries (AS/400) configuration (`iseries.zad`), navigates to the user's job log, and writes the contents of the job log to a file and store it in the following location:

`C:\JOBLOG.TXT`

To modify and use the script:

1. Create a new text document. Name it anything you want and save the file with a `.vbs` file extension.
2. Copy and paste the entire script into the document you just created.
3. Copy the script into the BlueZone `\scripts` folder.
4. Launch the BlueZone Script Host and Debugger program.
5. Click **File** ® **Open** from the menu bar.
6. Locate the script you just created and highlight it and click **Open**. The script appears in the main window.

```
' Sample Visual Basic script using BlueZone Host Automation Object
'
' This sample script will run a BlueZone session and then
' logon to an iSeries host and write the Job Log to the disk
' file C:\JOBLOG.TXT.
'

' Subroutine Main
'
Sub Main

' Instantiate a handle to BHAO
'
Set Host = CreateObject( "BZWhll.WhllObj" )

' Run BlueZone iSeries Display
' with session ID of S1
' use an exiting config file called iseries.zad
' timeout and return error if no signon screen after 30 seconds
' continue with script execution after host sends 1 screen paint
'
ResultCode = Host.OpenSession( 1, 1, "iseries.zad", 30, 1 )
If ResultCode <> 0 Then
   Host.MsgBox "Error connecting to host!", 4096
End If

' connect to session with hllapi id of "A"
' return error if session not found
'
ResultCode = Host.Connect( "A" )
If ResultCode <> 0 Then
   Host.MsgBox "Error connecting to session A!", 4096
End If

' logon to host
' then wait for host to unlock the keyboard
'
Host.SendKey "username@Tpassword@E"
Host.WaitReady 10, 1

' go to Display Job Log screen
Host.SendKey "1@E"
Host.WaitReady 10, 1
Host.SendKey "1@E"
```

```
Host.WaitReady 10, 1
Host.SendKey "10@E"
Host.WaitReady 10, 1

' create disk file C:\JOBLOG.TXT
'
Set fso = CreateObject( "Scripting.FileSystemObject" )
Set f = fso.OpenTextFile( "c:\joblog.txt", 2, True )

' write the job log screens to disk
' read the screens until "Bottom" is found at position
' row 19, column 74
'
MoreText = "More.."
BottomText = "Bottom"
While MoreText <> BottomText
   For i = 1 to 24
      Host.ReadScreen Buf, 80, i, 1
      f.WriteLine Buf
   Next
   f.WriteLine " "
   Host.ReadScreen MoreText, 6, 19, 74
Wend

' close the file
'
f.Close

' logoff from host
'
Host.SendKey "@E"
Host.WaitReady 10, 1
Host.SendKey "@3"
Host.WaitReady 10, 1
Host.SendKey "90@E"
Host.WaitReady 10, 1

' close BlueZone iSeries Display
' having session id of S1
'
Host.CloseSession 1, 1

' end of Subroutine Main
'
End Sub

' run Subroutine Main
'
Main
```

## Sample JScript

This script is designed to be used with BlueZone for the desktop. The script launches an already existing BlueZone iSeries (AS/400) configuration called `iseries.zad`, navigates to the end user's job log, and writes the contents of the job log to a file and store it in the following location:

`C:\JOBLOG.TXT`

To modify and use the script:

1. Create a text document and name it anything you want. Save the file with a `.js` extension.
2. Copy and paste the entire script into the document you just created.
3. Copy the script into the BlueZone `\scripts` folder.

4. Launch the BlueZone Script Host and Debugger program.
5. Select **File** ® **Open** from the menu bar.
6. Locate the script you just created and highlight it and click **Open**.
   The script appears in the BlueZone Script Host and Debugger main window.
7. Create a BlueZone configuration that connects to your iSeries host and be sure to enable the DDE interface.
8. Name the configuration `iseries.zad`.
9. Replace `username` and `password` in the script below with a valid user name and password.

```
// Sample JScript using BlueZone Host Automation Object

// This sample script will run a BlueZone session and then
// logon to an iSeries host and write the Job Log to the disk
// file C:\JOBLOG.TXT.

// Instantiate a handle to BlueZone Host Automation Object
host = new ActiveXObject( "BZWhll.WhllObj" )

// Run BlueZone iSeries Display with session ID of S1
// Timeout and return error if no signon screen after 30 seconds
// Continue with script execution after host sends 1 screen paint

ResultCode = host.OpenSession( 1, 1, "warrenton.zad", 30, 1 )
if ( ResultCode != 0 )
   host.MsgBox( "Error launching BlueZone session!", 4096 )

// connect to session with hllapi id of "A"
// return error if session not found

else

ResultCode = host.Connect( "A" )
if ( ResultCode != 0 )
   host.MsgBox( "Error connecting to session A!", 4096 )

// logon to host
// then wait for host to unlock the keyboard

host.SendKey( "username@Tpassword@E" )
host.WaitReady( 10, 1 )
// go to Display Job Log screen
host.SendKey( "1@E" )
host.WaitReady( 10, 1 )
host.SendKey( "1@E" )
host.WaitReady( 10, 1 )
host.SendKey( "10@E" )
host.WaitReady( 10, 1 )

// create disk file C:\JOBLOG.TXT
var ForReading = 1, ForWriting = 2, ForAppending = 3
var TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0
var fs, f, ts

// create the joblog file
fs = new ActiveXObject( "Scripting.FileSystemObject" )
fs.CreateTextFile( "c:\\joblog.txt" )
f = fs.GetFile( "c:\\joblog.txt" )
ts = f.OpenAsTextStream( ForWriting, TristateUseDefault )

// read the screens until "More..." is NOT found at position row 19, column 74
var MoreText = "More.."
var BottomText = "Bottom"
var Buf = new Object( )
```

```
if ( MoreText != BottomText )
    host.ReadScreen( Buf, 1920, 1, 1 )

// write the contents of the buffer to the screen
//    host.MsgBox( Buf.Str, 64 )               // optional message box

// write the contents of the buffer to disk
    ts.Write( Buf.Str )
    ts.Write( " " )
    host.ReadScreen( MoreText, 6, 19, 74 )

// close the file
ts.Close( )

// logoff from host
host.SendKey( "@E" )
host.WaitReady( 10, 1 )
host.SendKey( "@3" )
host.WaitReady( 10, 1 )
host.SendKey( "90@E" )
host.WaitReady( 10, 1 )

// close BlueZone iSeries Display having session id of S1
host.CloseSession( 1, 1 )
```

## Sample VBScript HTML

The following script is designed to be used with BlueZone Web-to-Host running in the embedded client mode. This script launches an embedded BlueZone IBM 3270 Mainframe display session from an Object Tag. Five buttons appear at the top of the web page. Once the session is launched, you can use the buttons to perform the following functions:

- **Log On**: Logs onto a TSO session and brings up the ISPF Menu.
- **Perform GetCursor**: Pops up a Message Box with the current Row and Column of the cursor.
- **Perform ReadScreen**: Pops up a Message Box with some text that is read from the screen.
- **Log Off**: Logs off the session.
- **Disconnect**: Disconnects the BlueZone Host Automation Object.

If you have an IBM 3270 Mainframe and you want to use the script as is, you must change the HostAddress value and possibly the Port value located in the Object Tag.

Also, you must change the `Username` and `Password` located in the `LogOn` function with your own valid user name and password.

This sample script also contains the `ScriptOnInitComplere` PARAM as shown here:

```
<PARAM NAME="ScriptLanguage" VALUE="VBScript">
<PARAM NAME="ScriptOnInitComplete" VALUE="hostConn">
```

The purpose of this PARAM is to delay the execution of your script until the BlueZone Web-to-Host control module and associated files are completely downloaded. To use it, create a Function that launches your script. Place that Function name in the PARAM VALUE as shown in the above example and in the sample script below.

To use the script:

1. Create a text document.
2. Save the file with any name you want and a file extension of `.htm` or `.html`.
3. Copy and paste the HTML code into the document you just created.

You can also cut and paste the script into an existing HTML page. Since this page contains the Object Tag, this page takes the place of your current Object Tag page.

**Note**

In order to use this sample HTML page and script, you must have BlueZone Web-to-Host installed and running on a web server with the optional BlueZone scripting cab file made available to the end users.

Refer to the "How to Include BlueZone Scripting Components" section in the *BlueZone Web-to-Host Administrator's Guide* for more information.

In order for the BlueZone Host Automation Object to communicate with a BlueZone Display emulation session, you must enable the BlueZone DDE interface. Refer to Configuring BlueZone, on page 47 for more information on enabling the DDE interface.

If you want the script to automatically log on to the host, uncomment the last two lines of the hostConn Function. The last two lines contain the words "else" and "logOn". By uncommenting these two lines, the function "hostConn" automatically calls the Function logOn.

## Differences between VBScript and JavaScript

There are a few syntax and usage differences between VBScript and JavaScript.

Refer to Sample JavaScript HTML, on page 93 for more information.

## Sample HTML page with VBScript

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>VBScript Sample Page</TITLE>

<SCRIPT language="VBScript">

Dim Host, Row, Col



' create Function hostConn which is called by "ScriptOnInitComplete" in the Object Tag

Function hostConn



  ' instantiate the BlueZone Object
  Set host = CreateObject( "BZWhll.WhllObj" )
  ResultCode = host.Connect( "A" )

  If ( ResultCode <> 0 ) Then

   ' display an error message if it can't connect
   host.MsgBox "Error connecting to session A!", 4096

  End If

End Function

' log on to the host
Function logOn

  host.Wait 1
  host.SendKey "TSO"
  host.SendKey "@E"
  host.Wait 1
```

```
  host.WaitReady 10, 1
  host.SendKey "username"
  host.SendKey "@E"
  host.WaitReady 10, 1
  host.SendKey "password"
  host.SendKey "@E"
  host.WaitReady 10, 1
  host.SendKey "@E"
  host.WaitReady 10, 2
  host.SendKey "@E"
  host.WaitReady 10, 1
  host.Focus

End Function

' *** GetCursor Example ***
Function getCursor

  host.GetCursor Row, Col
  host.MsgBox "The Row = "& Row & " , the Column = "& Col, 4096
  host.Focus

End Function

' *** ReadScreen Example ***
Function readScreen

  host.ReadScreen Buf, 8, 6, 19
  host.MsgBox "Buf is "& Buf, 4096
  host.Focus

End Function

' log off from host
Function logOff

  host.SendKey "@3"
  host.WaitReady 10, 1
  host.SendKey "logoff@E"
  host.WaitReady 10, 1
  host.focus
  host.CloseSession 0, 1
  host.Focus

End Function

' disconnect the BlueZone Object
Function disconnObj

  host.MsgBox "BZHAO Disconnected!", 4096
  host.disconnect

End Function

</SCRIPT>
</HEAD>
<BODY>

<DIV Style="Position:Absolute;Left:10px;Top:50px">

  <OBJECT ID="BlueZone Web-to-host control module v5"
  CLASSID="clsid:037790A6-1576-11D6-903D-00105AABADD3"
  CODEBASE="../controls/sglw2hcm.ocx#Version=-1,-1,-1,-1"
  HEIGHT=480
  WIDTH=740
```

```
  >
  <PARAM NAME="IniFile" VALUE="default.ini">
  <PARAM NAME="Sessions" VALUE="MD_S1">
  <PARAM NAME="DistFile" VALUE="default.dst">
  <PARAM NAME="MD_S1" VALUE="mainframe.zmd">
  <PARAM NAME="MD_S1_Save" VALUE="Yes">
  <PARAM NAME="ScriptLanguage" VALUE="VBScript">
  <PARAM NAME="ScriptOnInitComplete" VALUE="hostConn">
  <PARAM NAME="MD_S1_RunInBrowser" VALUE="Position">
  </OBJECT>

</DIV>

<FORM>
  <INPUT NAME="submit" TYPE=Button VALUE="Log On" onClick="logOn">
  <INPUT NAME="submit" TYPE=Button VALUE="Perform GetCursor" onClick="getCursor">
  <INPUT NAME="submit" TYPE=Button VALUE="Perform ReadScreen" onClick="readScreen">
  <INPUT NAME="submit" TYPE=Button VALUE="Log Off" onClick="logOff">
  <INPUT NAME="submit" TYPE=Button VALUE="Disconnect" onClick="disconnObj">
</FORM>

</SCRIPT>
</BODY>
</HTML>
```

## Sample JavaScript HTML

The following script is designed to be used with BlueZone Web-to-Host running in the Embedded Client Mode. This script launches an "embedded" BlueZone IBM 3270 Mainframe Display session from an Object Tag. Five buttons appear at the top of the web page. Once the session is launched, you can use the buttons to perform the following functions:

- **Log On**: Logs onto a TSO session and brings up the ISPF Menu
- **Perform GetCursor**: Pops up a Message Box with the current Row and Column of the cursor
- **Perform ReadScreen**: Pops up a Message Box with some text that is read from the screen
- **Log Off**: Logs off the session
- **Disconnect**: Disconnects the BlueZone Host Automation Object

If you have an IBM 3270 Mainframe and you want to use the script as is, you must change the HostAddress value and possibly the Port value located in the Object Tag.

Also, you must change the Username and Password located in the logOn() function with your own valid user name and password.

The sample also contains the `ScriptOnInitComplere` PARAM shown here:

```
<PARAM NAME="ScriptLanguage" VALUE="JavaScript">
<PARAM NAME="ScriptOnInitComplete" VALUE="hostConn()">
```

The purpose of this PARAM is to delay the execution of your script until the BlueZone Web-to-Host control module and associated files are completely downloaded. To use it, create a Function that launches your script. Place that Function name in the PARAM VALUE as shown in the above example and in the sample script below.

To use the script:

1. Create a text document.
2. Save the file with any name you want and a file extension of `.htm` or `.html`.
3. Copy and paste the HTML code into the document you just created.
   You can also cut and paste the script into an existing HTML page. Either way, since this page contains the Object Tag, this page takes the place of your current Object Tag page.

**Note**

In order to use this sample HTML page and script, you must have BlueZone Web-to-Host installed and running on a web server with the optional BlueZone scripting cab file made available to the end users.

Refer to the "How to Include BlueZone Scripting Components" section in the *BlueZone Web-to-Host Administrator's Guide* for more information. Also, In order for the BlueZone Host Automation Object to communicate with a BlueZone Display emulation session, you must enable the BlueZone DDE interface. Refer to Configuring BlueZone, on page 47 for more information on enabling the DDE interface.

If you want the script to automatically Log On to the host, uncomment the last two lines of the hostConn() function. The last two lines contain the words "else" and "logOn()". By uncommenting these two lines, the function "hostConn() automatically calls the function logOn().

## Differences Between VBScript and JavaScript

Some scripting languages such as JScript and JavaScript only support passing variable parameters to functions as Values (copies), as opposed to passing variable parameters to functions as References (pointers). In these scripting environments, parameters passed to functions that return data need to be of type "Object". The BlueZone Host Automation Object has built-in support for passing variables of type Object to its methods that return data, and uses the reserved object members "Str" and "Num" to "fill-in" the data to be returned. In the following Sample HTML Page, there are examples of the ReadScreen and Get Cursor Methods using correct JavaScript syntax.. The ReadScreen Method returns a string of text so "Str" has to be used, and the GetCursor Method returns a number value so "Num" has to be used.

In the BlueZone Object Methods Listing, on page 48, the examples are given in VBScript. However, when the syntax required by JavaScript is different than VBScript, the example is given in both VBScript and JavaScript.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>JavaScript Sample Page</TITLE>

<SCRIPT language="JavaScript" type="text/javascript">

// create function hostConn() which is called by "ScriptOnInitComplete" in the Object Tag
function hostConn() {

  // instantiate the BlueZone Object
  host = new ActiveXObject( "BZWhll.WhllObj" );
  ResultCode = host.Connect( "A" );

  if ( ResultCode != 0 )
  //display an error message if can't connect
  host.MsgBox( "Error connecting to session A!", 48 );
}

// log on to the host
function logOn() {
  host.SendKey( "TSO" );
  host.SendKey( "@E" );
  host.Wait( 1 );
  host.WaitReady( 10, 1 );
  host.SendKey( "username" );
  host.SendKey( "@E" );
  host.WaitReady( 10, 1 );
  host.SendKey( "password" );
  host.SendKey( "@E" );
  host.WaitReady( 10, 1 );
```

```
  host.SendKey( "@E" );
  host.WaitReady( 10, 2 );
  host.SendKey( "@E" );
  host.WaitReady( 10, 1 );
  host.Focus();
}

// *** GetCursor Example ***
function getCursor() {
  var Row = new Object();
  var Col = new Object();
  host.GetCursor( Row, Col );
  alert( "The Row = " + Row.Num + ", the Column = " + Col.Num );
  host.Focus();
}

// *** ReadScreen Example ***
function readScreen() {
  var Buf = new Object();
  host.ReadScreen (Buf, 8, 6, 19);
  alert( "The contents of Buf = " + Buf.Str );
  host.Focus();
}

// *** ReadScreen Example ***
function readScreen() {
  var Buf = new Object();
  host.ReadScreen (Buf, 8, 6, 19);
  alert( "The contents of Buf = " + Buf.Str );
  host.Focus();
}

// disconnect the BlueZone Object
function disconnObj() {
  alert ( "BZHAO Disconnected!" );
  host.disconnect();
}

</SCRIPT>
</HEAD>
<BODY>

<DIV Style="Position:Absolute;Left:10px;Top:50px">

  <OBJECT ID="BlueZone Web-to-host Control Module v5"
  CLASSID="clsid:037790A6-1576-11D6-903D-00105AABADD3"
  CODEBASE="../controls/sglw2hcm.ocx#Version=-1,-1,-1,-1"
  HEIGHT=480
  WIDTH=740
  >
  <PARAM NAME="IniFile" VALUE="default.ini">
  <PARAM NAME="Sessions" VALUE="MD_S1">
  <PARAM NAME="DistFile" VALUE="default.dst">
  <PARAM NAME="MD_S1" VALUE="mainframe.zmd">
  <PARAM NAME="MD_S1_Save" VALUE="Yes">
  <PARAM NAME="ScriptLanguage" VALUE="JavaScript">
  <PARAM NAME="ScriptOnInitComplete" VALUE="hostConn()">
  <PARAM NAME="MD_S1_RunInBrowser" VALUE="Position">
  </OBJECT>

</DIV>

<FORM>
  <INPUT NAME="submit" TYPE=Button VALUE="Log On" onClick="logOn()">
  <INPUT NAME="submit" TYPE=Button VALUE="Perform GetCursor" onClick="getCursor()">
```

```
  <INPUT NAME="submit" TYPE=Button VALUE="Perform ReadScreen" onClick="readScreen()">
  <INPUT NAME="submit" TYPE=Button VALUE="Log Off" onClick="logOff()">
  <INPUT NAME="submit" TYPE=Button VALUE="Disconnect" onClick="disconnObj()">
</FORM>

</SCRIPT>
</BODY>
</HTML>
```

## Connecting invisibly

If you are using an application to access host information through the BlueZone Host Automation Object, but you do not want to see the actual BlueZone session during the host access process, you can connect to the BlueZone Host Automation Object and set the Window to invisible.

This method works especially well if you have written a Windows application (like Visual Basic or PowerBuilder), that at some point requires a telnet connection to a host in order to retrieve information that can be used by the Windows application, but does not need or want the BlueZone emulation application to be visible to the user of the Windows application.

Set the BlueZone application Window's visibility to `False`:

```
Set Host = CreateObject( "BZWhll.WhllObj" )
Host.Connect "A"
Set wnd = Host.Window()
wnd.Visible = False
```

## Allowing multiple connections

It is possible to allow multiple connection requests from a single application/thread to the same session as long as it is using the same HLLAPI Short Name Session Identifier.

In order for BlueZone to not return an error, the **Allow Multiple Simultaneous Connections** check box must be enabled in the API Properties window.

Refer to *BlueZone Display and Printer Help* for more information on configuring BlueZone to work with HLLAPI applications.

# Multiple side-by-side versions of BlueZone

Starting with BlueZone Version 6.1, when invoking the BlueZone Host Automation objects the specific version can be specified in the `CreateObject()` statement. The following example loads Version 6.1 of the BZHAO:

```
set bzhao = CreateObject("BZWhll.WhllObj.6.1")
```

The `AllowCrossVersionInterop=` setting allows multiple versions of BlueZone to communicate and interact. This setting is defined in the `global.ini` file.

If `AllowCrossVersionInterop=Yes`, then the object can be used to communicate with any version of BlueZone. For example, `.Connect( "A" )` will successfully link the object to a different version session that has "`A`" configured as its session identifier.

If `AllowCrossVersionInterop=No`, then the object can only be used to communicate with BlueZone Version 6.1 sessions, the `.Connect( "A" )` method would then return `error 1 (Not Connected)`.

If running multiple versions of BlueZone, and the version is omitted in the `CreateObject()` statement, then the latest registered version of the BZHAO is used.

**Note**

Specifying the version in a script will tie the script to the particular version. If BlueZone is later upgraded to a newer version, then the script file must be modified to load the new object.

For more information on the `AllowCrossVersionInterop=` setting, refer to the *BlueZone Administrator's Guide* or the *BlueZone Web-to-Host Administrator's Guide* for more information.

# Chapter 5: BlueZone Basic

BlueZone Basic is a Visual Basic for Applications VBA and VBScript compatible Basic Scripting Language which can be used to add functionality to the BlueZone family of terminal emulation clients or Web pages to automate complex tasks. Users can also create scripts for launching and manipulating other applications via OLE Automation or external DLLs. BlueZone Basic is a complete programming language.

BlueZone Basic also supports the VBScript interface so that customers who are using VBScript can upgrade with no source code changes. BlueZone Basic is the complete solution for your scripting language needs.

BlueZone Basic supports a substantial subset of Visual Basic for Applications. In addition, BlueZone Basic supports Microsoft Word Dynamic Dialogs, offers full OLE support, fully interacts with Dynamic HTML forms, and has a powerful API that is easy to learn and easy to integrate.

Through the BlueZone Basic API you can share variables, objects, and functions with your application. BlueZone Basic scripts can get and set properties and call methods or objects in your application. Callbacks can be registered so that your application can respond to compile and runtime events such as undefined variables, functions and data types. In addition, BlueZone Basic has the ability to save and load compiled scripts, supports UNICODE, and can handle huge scripts (all with a footprint of about 400k).

End users can call functions that are embedded in DLLs or in the calling EXE. These functions must first be declared using the hidden but standard Declare Statement syntax. By building a string containing the declare statements and appending it prior to executing the user's script, these functions will appear to be built into the BlueZone Basic language and can be used on demand by your end users. New functions and subroutines can be added or redirected at run time though BlueZone Basic's powerful API.

### Features

- Adds functionality to the BlueZone family of Terminal Emulation products
- Uses the VBA standard for scripting
- Includes a powerful well designed API
- Easy to learn -- easy to use
- Scripts can even be written in VB first, then run with BlueZone Basic
- Supports a substantial subset of Microsoft Visual Basic for Applications
- Completely syntax compatible
- Complete Automation and ActiveX Support
- Works with BlueZone Script Host and Debugger

## Scripting language elements

This section describes the general elements of the BlueZone Basic language. BlueZone Basic scripts can include comments, statements, various representations of numbers, eleven (11) variable data types including user defined types, and multiple flow of control structures. BlueZone Basic is also extendable by calling external DLLs or calling functions back in the application's executable file.

# Comments

Comments are non-executed lines of code which are included for the benefit of the programmer. Comments can be included virtually anywhere in a script. Any text following an apostrophe or the word `Rem` is ignored by BlueZone Basic. `Rem` and all other keywords and most names in BlueZone Basic are not case sensitive.

```
'     This whole line is a comment
rem   This whole line is a comment
REM   This whole line is a comment
Rem   This whole line is a comment
```

Comments can also be included on the same line as executed code:

```
MsgBox Msg   ' Display message.
```

Everything after the apostrophe is a comment.

# Statements

In BlueZone Basic there is no statement terminator. More than one statement can be put on a line if they are separated by a colon.

For example:

```
X.AddPoint( 25, 100) : X.AddPoint(  0,  75)
```

Is equivalent to:

```
X.AddPoint( 25, 100)
X.AddPoint(  0,  75)
```

## Line continuation character

The underscore is the line continuation character in BlueZone Basic. There must be a space before and after the line continuation character:

```
X.AddPoint _
( 25, 100)
```

# Numbers

BlueZone Basic supports three representations of numbers:

- Decimal (default)
- Octal
- Hexadecimal

Most of the numbers used in this reference guide are expressed in decimal or base 10 format. However, if you need to use Octal (base 8) or hexadecimal (base 16) numbers, prefix the number with `&O` or `&H` respectively.

# Variable and constant names

Variable and constant names must begin with a letter. They can contain the letters A to Z and a to z, the underscore symbol (_), and the digits 0 to 9. Variable and constant names must begin with a letter, be no longer than 40 characters, and cannot be reserved words.

For a table of reserved words, refer to Functions, statements and reserved words - Quick reference, on page 122.

**Note**

One exception to this rule is that object member names and property names can be reserved words.

# Variable types

**Variant**

Like Visual Basic, when a variable is introduced in BlueZone Basic, it is not necessary to declare it first (see option explicit for an exception to this rule). When a variable is used but not declared then it is implicitly declared as a variant data type. Variants can also be declared explicitly using "As Variant" as in Dim x As Variant. The variant data type is capable of storing numbers, strings, dates, and times. When using a variant you do not have to explicitly convert a variable from one data type to another. This data type conversion is handled automatically.

For example:

```
Sub Main
   Dim x   ' Variant variable.
      x = 10
      x = x + 8
      x = "F" & x
   print x   ' Prints F18.
End Sub
```

A variant variable can readily change its type and its internal representation can be determined by using the function VarType. VarType returns a value that corresponds to the explicit data types. Refer to VarType function, on page 198 for more information.

When storing numbers in variant variables, the data type used is always the most compact type possible. For example, if you first assign a small number to the variant it is stored as an integer. If you then assign your variant to a number with a fractional component, it is then stored as a double.

For numeric operations on a variant variable, it is sometimes necessary to determine if the value stored is a valid numeric, thus avoiding an error. This can be done with the IsNumeric function. Refer to IsNumeric function, on page 165 for more information.

**Variants and Concatenation**

If a string and a number are concatenated the result is a string. To be sure your concatenation works regardless of the data type involved, use the & operator. The & operator does not perform arithmetic on your numeric values, it concatenates them as if they were strings.

The IsEmpty function can be used to find out if a variant variable has been previously assigned. Refer to IsEmpty function, on page 165 for more information.

# Other data types

Table 4: Data types lists the twelve data types available in BlueZone Basic.

Table 4: Data types

| Variable | Type definition | Size |
|---|---|---|
| Byte | Dim BVar As Byte | 0 to 255 |
| Boolean | Dim BoolVar As Boolean | True or False |
| String $ | Dim Str_Var As String | 0 to 65,500 char |
| Integer % | Dim Int_Var As Integer | 2 bytes |
| Long & | Dim Long_Var As Long | 4 bytes |
| Single ! | Dim Sing_Var As Single | 4 bytes |
| Double # | Dim Dbl_Var As Double | 8 bytes |
| Variant | Dim X As Any | |
| Currency | Dim Cvar As Currency | 8 bytes |
| Object | Dim X As Object | 4 bytes |
| Date | Dim D As Date | 8 bytes |
| User Defined Types | | size of each element |

### Scope of variables

BlueZone Basic scripts can be composed of many files and each file can have many subroutines and functions in it. Variable names can be reused even if they are contained in separate files. Variables can be local or global.

### Declaration of variables

In BlueZone Basic, variables are declared with the Dim statement. To declare a variable other than a variant, the variable must be followed by As, or appended by a type declaration character such as a % for Integer type.

For example:

```
Sub Main
   Dim X As Integer
   Dim Y As Double
   Dim  Name$, Age%   ' Multiple declaration on one line.
End Sub
```

## Global variables

Global variables are declared outside any subroutine or function using the Dim or Global statements and are accessible throughout the entire script. In addition, when a script ends execution, Global variables are stored in the `Bzshvars.ini` file, located in the BlueZone working directory. Subsequent running of scripts that use referenced Global variables have their values initialized to the stored values before script execution. This allows for variable sharing across multiple instances of the script host. A Global variable must be initialized in the script if it does not want to use the stored value.

Global variables are cleared when the last instance of a BlueZone emulator session is closed, or if the `Bzshvars.ini` file is deleted.

For example:

```
Dim x As Integer
Sub Main
   MsgBox x
   x = 10
End Sub
```

A message box displays 0 the first time the script is run and 10 on subsequent running of the script.

## $Include: statement

The $Include: statement can be used to insert script text contained in a separate file into the script memory block when playing and debugging scripts. This is useful if multiple scripts contain the same code. The $Include: statement must be preceded by the comment (') character. To comment-out an $Include: statement, use a comment at the beginning of the script line that is not part of the $Include: statement.

Example:

```
Sub Main
    x = 1
    '$Include: "c:\file1.bbh"
'   '$Include: "c:\file2.bbh"
End Sub
```

In the above example, text contained in file `C:\file1.bbh` is inserted into the script memory block when the script is run. The file `C:\file2.bbh` is not inserted because it is commented-out.

## Control structures

BlueZone Basic has complete process control functionality. The control structures available are Do loops, While loops, For loops, Select Case, If Then, and If Then Else. In addition, BlueZone Basic has one branching statement: GoTo. The Goto statement branches to the label specified in the Goto statement.

Example:

```
Goto label1

   .
   .
   .

label1:
```

The program execution jumps to the part of the program that begins with the label `label1:`.

### Do loops

The Do...Loop allows you to execute a block of statements an indefinite number of times. The variations of the Do...Loop are Do While, Do Until, Do Loop While, and Do Loop Until.

Example:

```
Do While|Until condition
   statement(s)...
   [Exit Do]
   statement(s)...
Loop

Do Until condition
   statement(s)...
Loop

Do
   statements...
Loop While condition

Do
   statements...
Loop Until condition
```

Do While and Do Until check the condition before entering the loop, thus the block of statements inside the loop are only executed when those conditions are met. Do Loop While and Do Loop Until check the condition after having executed the block of statements thereby guaranteeing that the block of statements is executed at least once.

## While loop

The While...Wend loop is similar to the Do While loop. The condition is checked before executing the block of statements comprising the loop.

Example:

```
While condition
   statements...
Wend
```

## For...Next loop

The For...Next loop has a counter variable and repeats a block of statements a set number of times. The counter variable increases or decreases with each repetition through the loop. The counter default is one if the Step variation is not used.

Example:

```
For counter = beginning value To ending value [Step increment]
   statements...
Next
```

## If and Select statements

The If...Then block has a single line and multiple line syntax. The condition of an If statement can be a comparison or an expression, but it must evaluate to True or False.

Example:

```
If condition Then statements...   ' Single line syntax.

If condition Then
   ' Multiple line syntax.
   statements...
End If
```

The other variation on the If statement is the If...Then...Else statement. Use this statement when there are different statement blocks to be executed depending on the condition. There is

also the If...Then...ElseIf... variation, these can get quite long and cumbersome, at which time consider using the Select statement.

Example:

```
If condition Then
   statements...
ElseIf condition Then
   statements...
Else

End If
```

The Select Case statement tests the same variable for many different values. This statement tends to be easier to read, understand, and follow and should be used in place of a complicated If...Then...ElseIf statement.

Example:

```
Select Case variable to test
   Case 1
      statements...
   Case 2
      statements...
   Case 3
      statements...
   Case Else
      statements...
End Select
```

Refer to Select Case statement, on page 186 for exact syntax and more code examples.

# Subroutines and functions

Subroutine and function names can contain the letters A to Z and a to z, the underscore symbol (_), and digits 0 to 9. The only limitation is that subroutine and function names must begin with a letter, be no longer than 40 characters, and not be reserved words.

Refer to Functions, statements and reserved words - Quick reference, on page 122 for a table of reserved words.

BlueZone Basic allows script developers to create their own functions or subroutines or to make DLL calls. Subroutines are created with the syntax `Sub <subname> ....  End Sub`. Functions are similar `Function <funcname> As <type> ...  <funcname> = <value> ... End Function`. DLL functions are declared via the Declare statement.

## ByRef and ByVal

ByRef gives other subroutines and functions the permission to make changes to variables that are passed in as parameters. The keyword ByVal denies this permission and the parameters cannot be reassigned outside their local procedure. ByRef is the BlueZone Basic default and does not need to be used explicitly. Because ByRef is the default all variables passed to other functions or subroutines can be changed, the only exception to this is if you use the ByVal keyword to protect the variable or use parentheses which indicate the variable is ByVal.

If the arguments or parameters are passed with parentheses around them, you tell BlueZone Basic that you are passing them ByVal. For example:

```
SubOne var1, var2, (var3)
```

The parameter var3 in this case is passed by value and cannot be changed by the subroutine SubOne:

```
Function R( X As String, ByVal n As Integer)
```

In the above example the function R is receiving two parameters X and n. The second parameter n is passed by value and the contents cannot be changed from within the function R.

In the following two code samples, scalar variable and user defined types are passed by reference.

## Scalar variables example

```
Sub Main
   Dim x(5) As Integer
   Dim i As Integer
   for i = 0 to 5
       x(i) = i
   next i
   Print i
   Joe (i), x   ' The parenthesis around it turn it into an expression
                ' which passes by value.
   print "should be 6: "; x(2), i
End Sub

Sub Joe( ByRef j As Integer, ByRef y() As Integer )
   print "Joe: "; j, y(2)
   j = 345
   for i = 0 to 5
   print "i: "; i; "y(i): "; y(i)
   next i
   y(2) = 3 * y(2)
End Sub
```

### Passing user defined types by ref to DLLs and BlueZone Basic functions example

```
' OpenFile() Structure
Type OFSTRUCT
   cBytes As String * 1
   fFixedDisk As String * 1
   nErrCode As Integer
   reserved As String * 4
   szPathName As String * 128
End Type

' OpenFile() Flags
   Global Const OF_READ = &H0
   Global Const OF_WRITE = &H1
   Global Const OF_READWRITE = &H2
   Global Const OF_SHARE_COMPAT = &H0
   Global Const OF_SHARE_EXCLUSIVE = &H10
   Global Const OF_SHARE_DENY_WRITE = &H20
   Global Const OF_SHARE_DENY_READ = &H30
   Global Const OF_SHARE_DENY_NONE = &H40
   Global Const OF_PARSE = &H100
   Global Const OF_DELETE = &H200
   Global Const OF_VERIFY = &H400
   Global Const OF_CANCEL = &H800
   Global Const OF_CREATE = &H1000
   Global Const OF_PROMPT = &H2000
   Global Const OF_EXIST = &H4000
   Global Const OF_REOPEN = &H8000

Declare Function OpenFile Lib "Kernel" (ByVal lpFileName As String, _
lpReOpenBuff As OFSTRUCT, ByVal wStyle As Integer) As Integer

Sub Main
   Dim ofs As OFSTRUCT
   ' Print OF_READWRITE
   ofs.szPathName = "c:\BlueZoneBasic\openfile.bbs"
   print ofs.szPathName
   ofs.nErrCode = 5
   print ofs.nErrCode
   OpenFile "t.bbs", ofs
   print ofs.szPathName
   print ofs.nErrCode
End Sub
```

## Calling procedures in DLLs

DLLs or Dynamic-link libraries are used extensively by engineers to access functions and subroutines located there. There are two main ways that BlueZone Basic can be extended:

- call functions and subroutines in DLLs
- call functions and subroutines located in the calling application

The mechanisms used for calling procedures in either place are similar.

Refer to the Declare statement, on page 138 for more details.

To declare a DLL procedure, or a procedure located in your calling application, place a declare statement in your declares file or outside the code area. All declarations in BlueZone Basic are Global to the run and accessible by all subroutines and functions. If the procedure does not return a value, declare it as a subroutine. If the procedure does have a return value, declare it as a function.

Example:

```
Declare Function GetPrivateProfileString Lib "Kernel32" _
(ByVal lpApplicationName As String, _
ByVal lpKeyName As String, ByVal lpDefault As String, _
  ByVal lpReturnedString As String, ByVal nSize _
As Integer, ByVal lpFileName As String) As Integer

Declare Sub InvertRect Lib "User" (ByVal hDC AS Integer, aRect As Rectangle)
```

In the above example, notice the use of the line extension character (_) the underscore. If a piece of code is too long to fit on one line, a line extension character can be used when needed.

Once a procedure is declared, you can call it just as you would any other BlueZone Basic function.

**Important**

BlueZone Basic cannot verify that you are passing correct values to a DLL procedure. If you pass incorrect values, the procedure can fail.

### Passing and returning strings

BlueZone Basic maintains variable-length strings internally as BSTRs. BSTRs are defined in the OLE header files as OLECHAR FAR *. An OLECHAR is a UNICODE character in 32-bit OLE and an ANSI character in 16-bit OLE. A BSTR can contain NULL values because a length is also maintained with the BSTR. BSTRs are also NULL terminated so they can be treated as an LPSTR. Currently this length is stored immediately prior to the string. This may change in the future, however, so you should use the OLE APIs to access the string length.

You can pass a string from BlueZone Basic to a DLL in one of two ways. You can pass it "by value" (ByVal) or "by reference". When you pass a string ByVal, BlueZone Basic passes a pointer to the beginning of the string data (that is, it passes a BSTR). When a string is passed by reference, BlueZone Basic passes a pointer to a pointer to the string data (that is, it passes a BSTR *).

### OLE API

SysAllocString/SysAllocStringLen

SysAllocString/SysAllocStringLen

SysFreeString

SysStringLen

SysReAllocStringLen

SysReAllocString

**Note**

The BSTR is a pointer to the string; you do not need to dereference it.

## File input/output

BlueZone Basic supports full sequential and binary file I/O.

The following functions and statements apply to file access:

- Dir
- EOF
- FileCopy
- FileLen

- Seek
- Close
- Input
- Line Input
- Print
- Write

### File I/O examples

```
Sub Main
   Open "TESTFILE" For Input As #1    ' Open file.
   Do While Not EOF(1)    ' Loop until end of file.
       Line Input #1, TextLine    ' Read line into variable.
           Print TextLine    ' Print to Debug window.
   Loop
   Close #1    ' Close file.
End Sub


Sub test

Open "MYFILE" For Input As #1    ' Open file for input.
Do While Not EOF(1)    ' Check for end of file.
   Line Input #1, InputData    ' Read line of data.
   MsgBox InputData
Loop
Close #1    ' Close file.

End Sub


Sub FileIO_Example()
   Dim Msg    ' Declare variable.
   Call Make3Files()    ' Create data files.
   Msg = "Several test files have been created on your disk."
   Msg = Msg & "Choose OK to remove the test files."
   MsgBox Msg
   For I = 1 To 3
      Kill "TEST" & I    ' Remove data files from disk.
   Next I
End Sub


Sub Make3Files ()
   Dim I, FNum, FName    ' Declare variables.
   For I = 1 To 3
       FNum = FreeFile    ' Determine next file number.
       FName = "TEST" & FNum
       Open FName For Output As FNum    ' Open file.
       Print #I, "This is test #" & I    ' Write string to file.
       Print #I, "Here is another ";  "line";  I
   Next I
   Close    ' Close all files.
End Sub
```

## Arrays

BlueZone Basic supports single- and multi-dimensional arrays. Using arrays you can refer to a series of variables by the same name each with a separate index. Arrays have upper and lower

bounds. BlueZone Basic allocates space for each index number in the array. Arrays should not be declared larger then necessary.

All the elements in an array have the same data type. BlueZone Basic supports arrays of bytes, Booleans, longs, integers, singles, double, strings, variants and user-defined types.

You can declare a fixed-size array with one of the following options:

- To create a global array, use the `Dim` statement outside the procedure section of a code module to declare the array.
- To create a local array, use the `Dim` statement inside a procedure.

BlueZone Basic supports Dynamic arrays.

To declare an array, the array name must be followed by the upper bound in parentheses. The upper bound must be an integer. For example:

```
Dim  ArrayName (10)  As Integer
Dim Sum (20) As Double
```

To create a global array, use `Dim` outside the procedure. For example:

```
Dim Counters (12)  As Integer
Dim Sums (26)  As Double
Sub Main () …
```

The same declarations within a procedure use `Static` or `Dim`. For example:

```
Static Counters (12)  As Integer
Static Sums (22)  As Double
```

The first declaration creates an array with 11 elements and with index numbers running from 0 to 10. The second creates an array with 21 elements. To change the default lower bound to 1, place an `Option Base` statement in the Declarations section of a module:

```
Option Base 1
```

Another way to specify the lower bound is to provide it explicitly (as an integer, in the range -32,768 to 32,767) using the To keyword:

```
Dim Counters (1 To 13)  As Integer
Dim Sums (100 To 126)  As String
```

In the preceding declarations, the index numbers of Counters run from 1 to 13 and the index numbers of Sums run from 100 to 126.

**Note**

Many other versions of Basic allow you to use an array without first declaring it. BlueZone Basic does not allow this; you must declare an array before using it.

Loops often provide an efficient way to manipulate arrays. For example, the following For loop initializes all elements in the array to 5:

```
Static Counters (1 To 20)  As Integer
   Dim I  As Integer
       For I = 1 To 20
           Counter ( I ) = 5
       Next I
…
```

### Multi-dimensional arrays

BlueZone Basic supports multidimensional arrays. For example the following example declares a two-dimensional array within a procedure:

```
Static Mat(20, 20) As Double
```

Either or both dimensions can be declared with explicit lower bounds:

```
Static Mat(1 to 10, 1 to 10) As Double
```

You can efficiently process a multidimensional array with the use of For loops. In the following statements the elements in a multi-dimensional array are set to a value:

```
Dim L As Integer, J  As Integer
   Static TestArray(1 To 10, 1 to 10)  As Double
     For L = 1 to 10
         For J = 1 to 10
             TestArray(L,J) = I * 10 + J
         Next J
     Next L
```

Arrays can be more than two-dimensional. BlueZone Basic does not have an arbitrary upper bound on array dimensions:

```
Dim ArrTest(5, 3, 2)
```

This declaration creates an array that has three dimensions with sizes 6 by 4 by 3 unless Option Base 1 is set previously in the code. The use of `Option Base 1` sets the lower bound of all arrays to 1 instead of 0.

## User-defined types

Users can define their own types that are composites of other built-in or user defined types. Variables of these new composite types can be declared and then member variables of the new type can be accessed using dot notation. Only variables of user defined types that contain simple data types can be passed to DLL functions expecting 'C' structures.

User-defined types are created using the type statement, which must be placed outside the procedure in your BlueZone Basic code. User defined types are global. The variables that are declared as user defined types can be either global or local. User-defined types in BlueZone Basic cannot contain arrays at this time.

Example:

```
Type type1
   a As Integer
   d As Double
   s As String
End Type

Type type2
   a As Integer
   o As type1
End Type

Dim type2a As type2
Dim type1a As type1

Sub TypeExample ()
   a = 5
   type1a.a = 7472
   type1a.d = 23.1415
   type1a.s = "YES"
   type2a.a = 43
   type2a.o.s = "Hello There"
   MsgBox type1a.a
   MsgBox type1a.d
   MsgBox type1a.s
   MsgBox type2a.a
   MsgBox type2a.o.s
   MsgBox a
End Sub
```

# Dialog support

BlueZone Basic has support for custom dialogs. The syntax is similar to the syntax used in Microsoft Word Basic. The dialog syntax is not part of Microsoft Visual Basic or Microsoft Visual Basic For Applications (VBA). BlueZone Basic has complete support for dialogs. The type of dialogs supported are outlined below.

## Dialog box controls

BlueZone Basic supports the standard Windows dialog box controls. This section introduces the controls available for custom dialog boxes and provides guidelines for using them.

The Dialog Box syntax begins with the statement `Begin Dialog`. The first two parameters of this statement are optional. If they are omitted the dialog is automatically centered.

Example:

```
Begin Dialog DialogName1 240, 184, "Test Dialog"

Begin Dialog DialogName1 60, 60,240, 184, "Test Dialog"
```

## OK and Cancel buttons

Every custom dialog box must contain at least one command button: an OK button or a Cancel button. BlueZone Basic includes separate dialog box definition statements for each of these two types of buttons.

Example:

```
Sub Main
   Begin Dialog ButtonSample 16,32,180,96,"OK and Cancel"
   OKButton 132,8,40,14
   CancelButton 132,28,40,14
End Dialog
Dim Dlg1 As ButtonSample
Button = Dialog (Dlg1)
End Sub
```

## List boxes, drop-down list boxes, and combo boxes

You can use a list box, drop-down list box, or combo box to present a list of items from which the user can select. A drop-down list box saves space (it can drop down to cover other dialog box controls temporarily). A combo box allows the user either to select an item from the list or type in a new item. The items displayed in a list box, drop-down list box, or combo box are stored in an array that is defined before the instructions that define the dialog box.

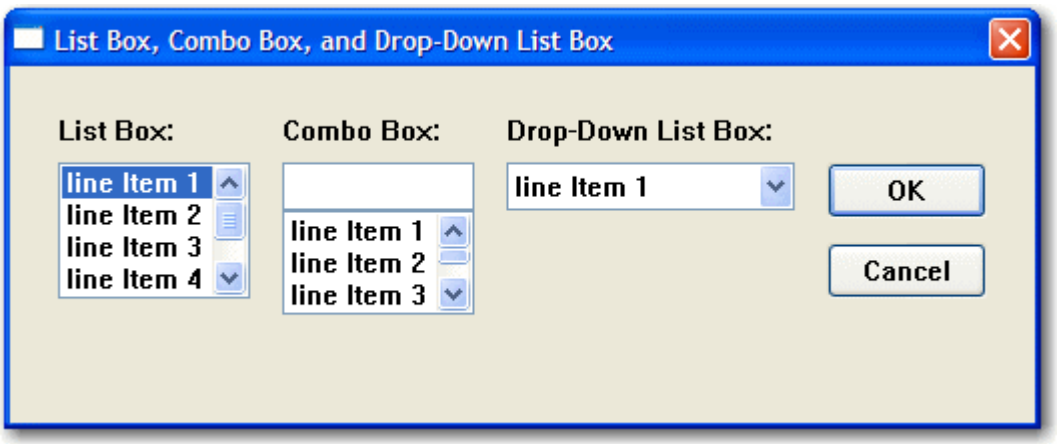
Example:

```
Sub Main
   Dim MyList$ (5)
   MyList (0) = "line Item 1"
   MyList (1) = "line Item 2"
   MyList (2) = "line Item 3"
   MyList (3) = "line Item 4"
   MyList (4) = "line Item 5"
   MyList (5) = "line Item 6"

Begin Dialog BoxSample 16,35,256,89,"List Box, Combo Box, and Drop-Down _
List Box"
   OKButton 204,24,40,14
   CancelButton 204,44,40,14
   ListBox 12,24,48,40, MyList$( ),.Lstbox
   DropListBox 124,24,72,40, MyList$( ),.DrpList
   ComboBox 68,24,48,40, MyList$( ),.CmboBox
   Text 12,12,32,8,"List Box:"
   Text 124,12,68,8,"Drop-Down List Box:"
   Text 68,12,44,8,"Combo Box:"
End Dialog
   Dim Dlg1 As BoxSample
   Button = Dialog ( Dlg1 )
End Sub
```

The above code produces the following dialog:

Figure 3: List box, combo box, and drop-down list box example



## Check boxes

You use a check box to make a yes or no, or on or off, choice. For example, you can use a check box to display or hide a toolbar in your application.

Example:

```
Sub Main
   Begin Dialog CheckSample15,32,149,96,"Check Boxes"
   OKButton 92,8,40,14
   CancelButton 92,32,40,14
   CheckBox 12,8,45,8,"CheckBox",.CheckBox1
   CheckBox 12,24,45,8,"CheckBox",.CheckBox2
   CheckBox 12,40,45,8,"CheckBox",.CheckBox3
   CheckBox 12,56,45,8,"CheckBox",.CheckBox4
End Dialog
   Dim Dlg1 As CheckSample
   Button = Dialog ( Dlg1 )
End Sub
```

## Text boxes and text

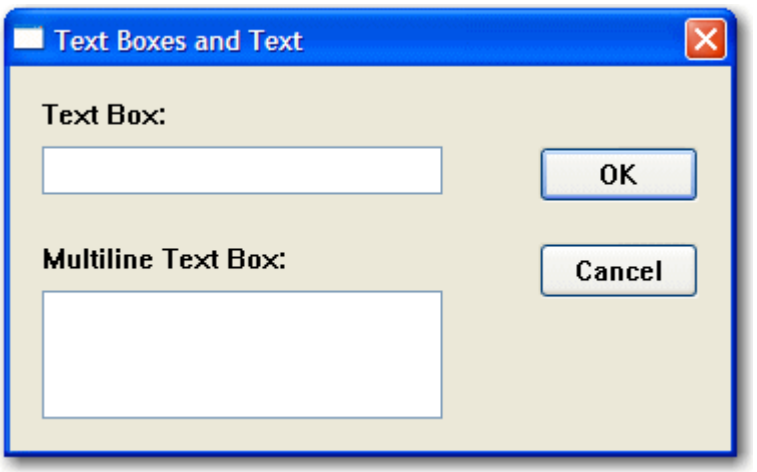
A text box control is a box in which the user can type text while the dialog box is displayed. By default, a text box holds a single line of text. BlueZone Basic supports single and multi-line text boxes. The last parameter of the TextBox function contains a variable to set the TextBox style.

Example:

```
Sub Main
   Begin Dialog TextBoxSample 16,30,180,96,"Text Boxes and Text"
   OKButton 132,20,40,14
   CancelButton 132,44,40,14
   Text 8,8,32,8,"Text Box:"
   TextBox 8,20,100,12,.TextBox1
   Text 8,44,84,8,"Multiline Text Box:"
   TextBox 8,56,100,32,.TextBox2
End Dialog
   Dim Dlg1 As TextBoxSample
   Button = Dialog ( Dlg1 )
End Sub
```

The above code produces the following dialog:

Figure 4: Text boxes and text example




The following is an additional example of how to implement a multi-line text box:

```
'=========================================================
' This sample shows how to implement a Multi-line Text Box
'=========================================================

' Try these different styles or-ed together as the last
' parameter of the TextBox changes the text box style.
' A 1 in the last parameter position defaults to a Multiline,
' Wantreturn, AutoVScroll text box.

Const ES_LEFT           = &h0000&
Const ES_CENTER         = &h0001&
Const ES_RIGHT          = &h0002&
Const ES_MULTILINE      = &h0004&
Const ES_UPPERCASE      = &h0008&
Const ES_LOWERCASE      = &h0010&
Const ES_PASSWORD       = &h0020&
Const ES_AUTOVSCROLL    = &h0040&
Const ES_AUTOHSCROLL    = &h0080&
Const ES_NOHIDESEL      = &h0100&
Const ES_OEMCONVERT     = &h0400&
Const ES_READONLY       = &h0800&
Const ES_WANTRETURN     = &h1000&
Const ES_NUMBER         = &h2000&

Sub Multiline
   Begin Dialog DialogType 60, 60, 140, 185, "Multiline text Dialog", _
   .DlgFunc
       TextBox 10, 10, 120, 150, .joe, ES_MULTILINE Or _
       ES_AUTOVSCROLL Or _
       ES_WANTRETURN   ' Indicates Multiline TextBox.
         'TextBox 10, 10, 120, 150, .joe, 1    ' Indicates Multiline TextBox.
         CancelButton 25, 168, 40, 12
     OKButton 75, 168, 40, 12
   End Dialog

   Dim Dlg1 As DialogType
   Dlg1.joe  = "The quick brown fox jumped over the lazy dog"
   ' Dialog returns -1 for OK, 0 for Cancel.
   Button = Dialog( Dlg1 )
   'MsgBox "button: " & button
   If button = 0 Then Exit Sub

   MsgBox "TextBox: "& Dlg1.joe
End Sub
```

## Option buttons and group boxes

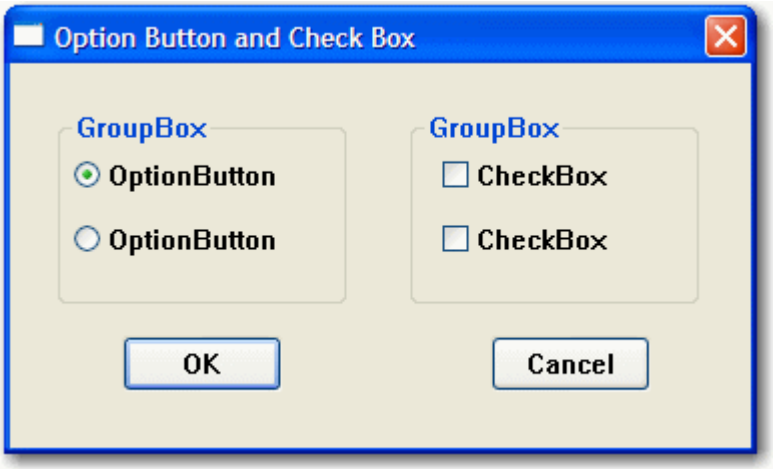
Option buttons allow the user to choose one option from several options. Typically, you use a group box to surround a group of option buttons, but you can also use a group box to set off a group of check boxes or any related group of controls.

Example:

```
Sub Main
   Begin Dialog GroupSample 31,32,185,96,"Option Button and Check Box"
       OKButton 28,68,40,14
           CancelButton 120,68,40,14
       GroupBox 12,12,72,48,"GroupBox",.GroupBox1
       GroupBox 100,12,72,48,"GroupBox",.GroupBox2
       OptionGroup .OptionGroup1
       OptionButton 16,24,54,8,"OptionButton",.OptionButton1
       OptionButton 16,40,54,8,"OptionButton",.OptionButton2
       CheckBox 108,24,45,8,"CheckBox",.CheckBox1
       CheckBox 108,40,45,8,"CheckBox",.CheckBox2
   End Dialog
       Dim Dlg1 As GroupSample
       Button = Dialog (Dlg1)
End Sub
```

The above code produces the following dialog:

Figure 5: Option button and check box example




## Dialog function

BlueZone Basic supports the dialog function. This function is a user-defined function that can be called while a custom dialog box is displayed. The dialog function makes nested dialog boxes possible and receives messages from the dialog box while it is still active.

When the function dialog() is called in BlueZone Basic, it displays the dialog box, and calls the dialog function for that dialog. BlueZone Basic calls the dialog function to see if there are any commands to execute. Typical commands that might be used are disabling or hiding a control. By default, all dialog box controls are enabled. If you want a control to be hidden you must explicitly make it disabled during initialization. After initialization BlueZone Basic displays the dialog box. When an action is taken by the user, BlueZone Basic calls the dialog function and passes values to the function that indicate the kind of action to take and the control that was acted upon.

The dialog box and its function are connected in the dialog definition. A function name argument is added to the `Begin Dialog` instruction, and matches the name of the dialog function located in your BlueZone Basic program. For example:

```
Begin Dialog UserDialog1 60,60, 260, 188, "3", .BlueZone Basic
```

## Dialog box control identifiers

A dialog function requires an identifier for each dialog box control that it acts on. The dialog function uses string identifiers. String identifiers are the same as the identifiers used in the dialog record. For example:

```
CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
```

The controls identifier and label are different. An identifier begins with a period and is the last parameter in a dialog box control instruction. In the sample code above `Check to display controls` is the label and `Chk1` is the identifier.

## Dialog function syntax

The syntax for the dialog function is as follows:

```
Function FunctionName( ControlID$, Action%, SuppValue% )
   Statement Block
   FunctionName = ReturnValue
End Function
```

A dialog function returns a value when the user chooses a command button. BlueZone Basic acts on the value returned. The default is to return 0 (zero) and close the dialog box. If a non zero is assigned the dialog box remains open. By keeping the dialog box open, the dialog function allows the user to do more than one command from the same dialog box.

**Note**

All parameters in the dialog function are required.

### Parameters

**`ControlID$`**
Receives the identifier of the dialog box control.

**`Action`**
Identifies the action that calls the dialog function. There are six possibilities, BlueZone Basic supports the first 4:

**Action 1**
The value passed before the dialog becomes visible.

**Action 2**
The value passed when an action is taken (for example, a button is pushed, check box is checked, and other actions). The **`controlID$`** is the same as the identifier for the control that was chosen.

**Action 3**
Corresponds to a change in a text box or combo box. This value is passed when a control loses the focus (for example, when the user presses the TAB key to move to a different control) or after the user clicks an item in the list of a combo box (an Action value of 2 is passed first). Note that if the contents of the text box or combo box do not change, an Action value of 3 is not passed. When Action is 3, **`ControlID$`** corresponds to the identifier for the text box or combo box whose contents were changed.

**Action 4**
Corresponds to a change of focus. When Action is 4, **`ControlID$`** corresponds to the identifier of the control that is gaining the focus. **`SuppValue`** corresponds to the numeric identifier for the control that lost the focus. A Dialog function cannot display a message box or dialog box in response to an Action value of 4.

**SuppValue**

Receives supplemental information about a change in a dialog box control. The information SuppValue receives depends on which control calls the dialog function. The following SuppValue values are passed when Action is 2 or 3.

Table 5: Passed SuppValue values

| Control | SuppValue passed |
|---|---|
| ListBox, DropListBox, or ComboBox | Number of the item selected where 0 (zero) is the first item in the list box, 1 is the second item, and so on. |
| CheckBox | 1 if selected, 0 (zero) if cleared. |
| OptionButton | Number of the option button selected, where 0 (zero) is the first option button within a group, 1 is the second option button, and so on. |
| TextBox | Number of characters in the text box. |
| ComboBox | If Action is 3, number of characters in the combo box. |
| CommandButton | A value identifying the button chosen. This value is not often used, since the same information is available from the ControlID$ value. |

## Statements and functions used in dialog functions

Table 6: Dialog statements and functions

| Statement or function | Action or result |
|---|---|
| DlgControlId | Returns the numeric equivalent of Identifier$, the string identifier for a dialog box control. |
| DlgEnable, DlgEnable() | The DlgEnable statement is used to enable or disable a dialog box control. When a control is disabled, it is visible in the dialog box, but is dimmed and not functional. DlgEnable() is used to determine whether or not the control is enabled. |
| DlgFocus, DlgFocus() | The DlgFocus statement is used to set the focus on a dialog box control. (When a dialog box control has the focus, it is highlighted.) DlgFocus() returns the identifier of the control that has the focus. |
| DlgListBoxArray, DlgListBoxArray() | The DlgListBoxArray statement is used to fill a list box or combo box with the elements of an array. It can be used to change the contents of a list box or combo box while the dialog box is displayed. DlgListBoxArray() returns an item in an array and the number of items in the array. |
| DlgSetPicture | The DlgSetPicture statement is used in a dialog function to set the graphic displayed by a picture control. |
| DlgText, DlgText | The DlgText statement is used to set the text or text label for a dialog box control. The DlgText() function returns the label of a control. |
| DlgValue, DlgValue() | The DlgValue statement is used to select or clear a dialog box control. Then DlgValue() function returns the setting of a control. |
| DlgVisible, DlgVisible() | The DlgVisible statement is used to hide or show a dialog box control. The DlgVisible() function is used to determine whether a control is visible or hidden. |

## DlgControlId function

```
DlgControlId(Identifier)
```

Used within a dialog function to return the numeric identifier for the dialog box control specified by Identifier, the string identifier of the dialog box control. Numeric identifiers are numbers, starting at 0 (zero) , that correspond to the positions of the dialog box control instructions within a dialog box definition. For example, consider the following instruction in a dialog box definition:

```
CheckBox 90, 50, 30, 12, , .MyCheckBox
```

The instruction `DlgControlId(MyCheckBox)` returns 0 (zero) if the CheckBox instruction is the first instruction in the dialog box definition, 1 if it is the second, and so on.

In most cases, your dialog functions perform actions based on the string identifier of the control that was selected.

## DlgFocus Statement, DlgFocus() function

```
DlgFocus Identifier
DlgFocus()
```

The `DlgFocus` statement is used within a dialog function to set the focus on the dialog box control identified by Identifier while the dialog box is displayed. When a dialog box control has the focus, it is active and responds to keyboard input. For example, if a text box has the focus, any text you type appears in that text box.

The `DlgFocus()` function returns the string identifier for the dialog box control that currently has the focus.

This example sets the focus on the control MyControl1 when the dialog box is initially displayed. (The main subroutine that contains the dialog box definition is not shown.)

```
Function MyDlgFunction( identifier, action, suppvalue )
Select Case action
   Case 1   ' The dialog box is displayed
      DlgFocus MyControl1
   Case 2   ' Statements that perform actions based on which control
            ' is selected.
   End Select
End Function
```

## DlgListBoxArray, DlgListBoxArray()

```
DlgListBoxArray Identifier, ArrayVariable()
DlgListBoxArray(Identifier, ArrayVariable())
```

The `DlgListBoxArray` statement is used within a dialog function to fill a ListBox, DropListBox, or ComboBox with the contents of `ArrayVariable()` while the dialog box is displayed.

The `DlgListBoxArray()` function fills `ArrayVariable()` with the contents of the ListBox, DropListBox, or ComboBox specified by Identifier and returns the number of entries in the ListBox, DropListBox, or ComboBox. The `ArrayVariable()` parameter is optional (and currently not implemented) with the `DlgListBoxArray()` function; if `ArrayVariable()` is omitted, `DlgListBoxArray()` returns the number of entries in the specified control.

## DlgSetPicture

```
DlgSetPicture  Identifier, PictureName
```

The `DlgSetPicture` function is used to set the graphic displayed by a picture control in a dialog.

The `Identifier` is a string or numeric representing the dialog box. The `PictureName` is a string that identifies the picture to be displayed.

## DlgValue, DlgValue()

```
DlgValue Identifier, Value
DlgValue(Identifier)
```

The `DlgValue` statement is used in a dialog function to select or clear a dialog box control by setting the numeric value associated with the control specified by Identifier. For example, `DlgValue MyCheckBox, 1` selects a check box, `DlgValue MyCHeckBox, 0` clears a check box, and `DlgValue MyCheckBox, -1` fills the check box with gray. An error occurs if Identifier specifies a dialog box control such as a text box or an option button that cannot be set with a numeric value.

The following dialog function uses a Select Case control structure to check the value of Action. The **`SuppValue`** is ignored in this function.

```
' This sample file outlines dialog capabilities, including nesting
' dialog boxes.
Sub Main
   Begin Dialog UserDialog1 60,60, 260, 188, "3", .DlgFunc
      Text 8,10,73,13, "Text Label:"
      TextBox 8, 26, 160, 18, .FText
       CheckBox 8, 56, 203, 16, "Check to display controls",. Chk1
       GroupBox 8, 79, 230, 70, "This is a group box:", .Group
       CheckBox 18,100,189,16, "Check to change button text", .Chk2
       PushButton 18, 118, 159, 16, "File History", .History
       OKButton 177, 8, 58, 21
       CancelButton 177, 32, 58, 21
   End Dialog

   Dim Dlg1 As UserDialog1
   x = Dialog( Dlg1 )
End Sub

Function DlgFunc( ControlID$, Action%, SuppValue%)

Begin Dialog UserDialog2 160,160, 260, 188, "3", .DlgFunc
   Text 8,10,73,13, "New dialog Label:"
   TextBox 8, 26, 160, 18, .FText
   CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
   CheckBox 18,100,189,16, "Additional CheckBox", .ch2
   PushButton 18, 118, 159, 16, "Push Button", .but1
   OKButton 177, 8, 58, 21
   CancelButton 177, 32, 58, 21
End Dialog
Dim Dlg2 As UserDialog2
Dlg2.FText = "Your default string goes here"

Select Case Action%

Case 1
   DlgEnable "Group", 0
   DlgVisible "Chk2", 0
   DlgVisible "History", 0
Case 2
   If ControlID$ = "Chk1" Then
       DlgEnable "Group"
       DlgVisible "Chk2"
       DlgVisible "History"
   End If

   If ControlID$ = "Chk2" Then
      DlgText "History", "Push to display nested dialog"
   End If

   If ControlID$ = "History" Then
       DlgEnable =1
       x = Dialog( Dlg2 )
   End If

Case Else

End Select
DlgEnable =1

End Function
```

# OLE Automation

OLE Automation is a standard, promoted by Microsoft, that applications use to expose their OLE objects to development tools, BlueZone Basic, and containers that support OLE Automation. A spreadsheet application can expose a worksheet, chart, cell, or range of cells all as different types of objects. A word processor can expose objects such as application, paragraph, sentence, bookmark, or selection.

When an application supports OLE Automation, the objects it exposes can be accessed by BlueZone Basic. You can use BlueZone Basic to manipulate these objects by invoking methods on the object, or by getting and setting the object's properties, just as you would with the objects in BlueZone Basic. For example, if you created an OLE Automation object named `MyObj`, you can write code such as this to manipulate the object:

```
Sub Main
   Dim MyObj As Object
   Set MyObj = CreateObject ("Word.Basic")
   MyObj.FileNewDefault
   MyObj.Insert "Hello, world."
   MyObj.Bold 1
End Sub
```

The following syntax is supported for the `GetObject` function:

```
Set MyObj = GetObject ("", class)
```

Where `class` is the parameter representing the class of the object to retrieve. The first parameter at this time must be an empty string.

The properties and methods an object supports are defined by the application that created the object. See the application's documentation for details on the properties and methods it supports.

## Accessing an object

The following functions and properties allow you to access an OLE Automation object.

Table 7: Functions and properties to access OLE Automation objects

| Name | Description |
|---|---|
| `CreateObject` | Function Creates a new object of a specified type. |
| `GetObject` | Function Retrieves an object pointer to a running application. |

# Scripting language overview

## Functions and statements - Quick reference

**Flow of control**

Goto, End, OnError, Stop, Do...Loop, Exit Loop, For...Next, Exit For, If..Then..Else...End If, Stop, While...Wend, Select Case

**Converting**

Chr, Hex, Oct, Str, CDbl, CInt, CIng, CSng, CStr, CVar, CVDate, Asc, Val, Date, DateSerial, DateValue, Format, Fix, Int, Day, Weekday, Month, Year, Hour, Minute, Second, TimeSerial, TimeValue

**Dialog**
Text, TextBox, ListBox, DropList, ComboBox, CheckBox, OKButton, BeginDialog, EndDialog, OptionGroup, OKButton, CancelButton, PushButton, Picture, GroupBox, Multi-line TextBox

**File I/O**
FileCopy, ChDir, ChDrive, CurDir, CurDir, MkDir,RmDir, Open, Close, Print #, Kill, FreeFile, LOF, FileLen, Seek, EOF, Write #, Input, Line Input, Dir, Name, GetAttr, SetAttr, Dir, Get, Put

**Math**
Exp, Log, Sqr, Rnd, Abs, Sgn, Atn, Cos, Sin, Tan, Int, Fix

**Procedures**
Call, Declare, Function, End Function, Sub, End Sub, Exit, Global

**Strings**
Let, Len, InStr, Left, Mid, Asc, Chr, Right, LCase, Ucase, InStr, LTrim, RTrim, Trim, Option Compare, Len, Space, String, StrComp Format

**Variables and constants**
Dim, IsNull, IsNumeric,VarType, Const, IsDate, IsEmpty, IsNull, Option Explicit, Global, Static

**Error trapping**
On Error, Resume

**Date/Time**
Date, Now, Time, Timer

**DDE**
DDEInitiate, DDEExecute, DDETerminate

**Arrays**
Option Base, Option Explicit, Static, Dim, Global, Lbound, Ubound, Erase, ReDim

**Miscellaneous**
SendKeys, AppActivate, Shell, Beep, Rem, CreateObject, GetObject, Randomize

## Functions, statements and reserved words - Quick reference

Abs, Access, Alias, And, Any

App, AppActivate, Asc, Atn, As

Base, Beep, Begin, Binary, ByVal

Call, Case, ChDir, ChDrive, Choose, Chr, Const, Cos, CurDir, CDbl, CInt, CLng, CSng, CStr, CVar, CVDate,Close, CreateObject

Date, Day, Declare, Dim, Dir, Do...Loop,Dialog, DDEInitiate

DDEExecute, DateSerial, DateValue, Double

Else, ElseIf, End, EndIf, EOF, Eqv, Erase, Err, Error

Exit, Exp, Explicit

False, FileCopy, FileLen, Fix, For

For...Next, Format, Function

Get, GetAttr, GoTo, Global, Get Object

Hex, Hour

If...Then...Else...[End If], Imp, Input, InputBox, InStr, Int, Integer, Is, IsEmpty, IsNull, IsNumeric, IsDate

Kill

LBound, LCase, Left, Len, Let, LOF,Log, Long, Loop, LTrim Line Input

Mid,Minute, MkDir, Mod, Month, MsgBox

Name, Next, Not, Now

Oct,On, Open, OKButton,Object, Option, Optional, Or, On Error

Print, Print #, Private, Put

Randomize, Rem, ReDim, RmDir, Rnd, Rtrim

Seek, SendKeys, Set, SetAttr, Second, Select, Shell, Sin, Sqr, Stop,Str, Sng, Single, Space, Static, Step, Stop, Str, String, Sub, StringComp

Tan,Text, TextBox, Time, Timer, TimeSerial, TimeVale, Then, Type, Trim, True, To, Type

UBound, UCase, Ucase, Until

Val, Variant, VarType

Write #, While, Weekday, Wend, With

Xor

Year

# Data types

Table 8: Data types

| Variable | Type specifier | Usage |
|---|---|---|
| String | $ | Dim Str_Var As String |
| Integer | % | Dim Int_Var As Integer |
| Long | & | Dim Long_Var As Long |
| Single | ! | Dim Sing_Var As Single |
| Double | # | Dim Dbl_Var As Double |
| Variant | | Dim X As Any |
| Boolean | | Dim X As Boolean |
| Byte | | Dim X As Byte |
| Object | | Dim X As Object |
| Currency | | (Not currently supported) |

# Operators

Table 9: Arithmetic operators lists the arithmetic operators.

Table 9: Arithmetic operators

| Operator | Function | Usage |
|---|---|---|
| ^ | Exponentiation | x = y^2 |
| - | Negation | x = -2 |
| * | Multiplication | x% = 2 * 3 |
| / | Division | x = 10/2 |
| Mod | Modulo | x = y Mod z |
| + | Addition | x = 2 + 3 |
| - | Subtraction | x = 6 - 4 |

Arithmetic operators follow mathematical rules of precedence.

**Note**

'+' or '&' can be used for string concatenation.

Table 10: Operator precedence lists the operator precedence from highest to lowest.

Table 10: Operator precedence

| Operator | Description |
|---|---|
| () | parenthesis (highest precedence) |
| ^ | exponentiation |
| - | unary minus |
| /,* | division/multiplication |
| mod | modulo |
| +, -, & | addition, subtraction, concatenation |
| =, <>, <, >,<=,>= | relational |
| not | logical negation |
| and | logical conjunction |
| or | logical disjunction |
| Xor | logical exclusion |
| Eqv | logical Equivalence |
| Imp | logical Implication (lowest precedence) |

Table 11: Relational operators lists the relational operators.

Table 11: Relational operators

| Operator | Function | Usage |
|---|---|---|
| < | Less than | x < Y |
| <= | Less than or equal to | x <= Y |
| = | Equals | x = Y |
| >= | Greater than or equal to | x >= Y |
| > | Greater than | x > Y |
| <> | Not equal to | x <> Y |

Table 12: Logical operators lists the logical operators.

Table 12: Logical operators

| Operator | Function | Usage |
|---|---|---|
| Not | Logical Negation | If Not (x) |
| And | Logical And | If (x> y) And (x < Z) |
| Or | Logical Or | if (x = y) Or (x = z) |

# Language reference

## Abs function

`Abs( number )`

Returns the absolute value of a number.

The data type of the return value is the same as that of the number argument. However, if the number argument is a Variant of `VarType (String)` and can be converted to a number, the return value is a Variant of `VarType (Double)`. If the numeric expression results in a Null, _Abs returns a Null.

### Example

```
Sub Main
   Dim Msg, X, Y

   X = InputBox("Enter a Number:")
   Y = Abs(X)

   Msg = "The number you entered is " & X
   Msg = Msg + ". The Absolute value of " & X & " is " & Y
   MsgBox Msg   'Display Message.
End Sub
```

## AppActivate statement

`AppActivate "app"`

Activates an application.

The parameter app is a string expression and is the name that appears in the title bar of the application window to activate.

### Example

```
Sub Main()
   AppActivate "Microsoft Word"
   SendKeys “%F,%N, BlueZone Basic”,True
   Msg = “Click OK to close Word”
   MsgBox Msg
   AppActivate “Microsoft Word”
   SendKeys “%F,%C,N”, True
End Sub
```

## Asc function

Asc( *str* )

Returns a numeric value that is the ASCII code for the first character in a string.

### Example

```
Sub Main()
   Dim I, Msg   ' Declare variables.
   For I = Asc("A") To Asc("Z")   ' From A through Z.
       Msg = Msg & Chr(I)   ' Create a string.
   Next I
   MsgBox Msg   ' Display results.
End Sub
```

## Atn function

Atn( *rad* )

Returns the arc tangent of a number.

The argument *rad* can be any numeric expression. The result is expressed in radians.

### Example

```
Sub AtnExample()
   Dim Msg, Pi   ' Declare variables.
   Pi = 4 * Atn(1)   ' Calculate Pi.
   Msg = "Pi is equal to " & Str(Pi)
   MsgBox Msg   ' Display results.
End Sub
```

## Beep statement

Beep

Sounds a tone through the computer's speaker. The frequency and duration of the beep depends on hardware, which can vary among computers.

### Example

```
Sub BeepExample()
   Dim Answer, Msg   ' Declare variables.
   Do
       Answer = InputBox("Enter a value from 1 to 3.")
       If Answer >= 1 And Answer <= 3 Then   ' Check range.
           Exit Do   ' Exit Do...Loop.
       Else
           Beep   ' Beep if not in range.
       End If
   Loop
   MsgBox "You entered a value in the proper range."
End Sub
```

## Call statement

`Call funcname [(parameter(s)]`

or

`[parameter(s)]`

Activates a BlueZone Basic subroutine called name or a DLL function with the name name. The first parameter is the name of the function or subroutine to call, and the second is the list of arguments to pass to the called function or subroutine.

You are never required to use the `Call` statement when calling a BlueZone Basic subroutine or a DLL function. Parentheses must be used in the argument list if the `Call` statement is being used.

### Example

```
Sub Main()
   Call Beep
   MsgBox "Returns a Beep"
End Sub
```

## CBool function

`CBool( expression )`

Converts expressions from one data type to a boolean. The parameter `expression` must be a valid string or numeric expression.

### Example

```
Sub Main
   Dim A, B, Check
   A = 5: B = 5
   Check = CBool(A = B)
   Print Check
   A = 0
   Check = CBool(A)
   Print Check
End Sub
```

## CDate function

`CVDate( expression )`

Converts any valid expression to a Date variable with a vartype of 7.

The parameter `expression` must be a valid string or numeric date expression and can represent a date from January 1, 30 through December 31, 9999.

### Example

```
Sub Main
   Dim MyDate, MDate, MTime, MSTime
   MybDate = "May 29, 1959"   ' Define date.
   MDate = CDate(MybDate)   ' Convert to Date data type.

   MTime = "10:32:27 PM"   ' Define time.
   MSTime = CDate(MTime)   ' Convert to Date data type.

   Print MDate
   Print MSTime
End Sub
```

## CDbl function

`CDbl( expression )`

Converts expressions from one data type to a double. The parameter `expression` must be a valid string or numeric expression.

### Example

```
Sub Main()
   Dim y As Integer
   y = 25555   ' The integer expression only allows for 5 digits.
   If  VarType(y) = 2 Then
   Print y

       x = CDbl(y)   ' Converts the integer value of y to a double
                     ' value in x.
       x = x * 100000   ' y is now 10 digits in the form of x.
       Print x
   End If
End Sub
```

## ChDir statement

`ChDir pathname`

Changes the default directory.

Pathname: `[drive:]  [ \ ] dir[\dir]...`

The parameter `pathname` is a string limited to fewer then 128 characters.

The `drive` parameter is optional. The `dir` parameter is a directory name. `ChDir` changes the default directory on the current drive, if the drive is omitted.

### Example

```
Sub Main()
   Dim Answer, Msg, NL   ' Declare variables.
   NL = Chr(10) ' Define newline.
   CurPath = CurDir()   ' Get current path.
   ChDir "\"
   Msg = "The current directory has been changed to "
   Msg = Msg & CurDir() & NL & NL & "Press OK to change back "
   Msg = Msg & "to your previous default directory."
   Answer = MsgBox(Msg)   ' Get user response.
   ChDir CurPath   ' Change back to user default.
   Msg = "Directory changed back to " & CurPath & "."
   MsgBox Msg   ' Display results.
End Sub
```

## ChDrive statement

`ChDrive` *`drivename`*

Changes the default drive.

The parameter `drivename` is a string and must correspond to a an existing drive. If `drivename` contains more than one letter, only the first character is used.

### Example

```
Sub Main()
   Dim Msg, NL   ' Declare variables.
   NL = Chr(10)   ' Define newline.
   CurPath = CurDir()   ' Get current path.
   ChDir "\"
   ChDrive "C:"
   Msg = "The current directory has been changed to "
   Msg = Msg & CurDir() & NL & NL & "Press OK to change back "
   Msg = Msg & "to your previous default directory."
   MsgBox Msg   ' Get user response.
   ChDir CurPath   ' Change back to user default.
   Msg = "Directory changed back to " & CurPath & "."
   MsgBox Msg   ' Display results.
End Sub
```

## CheckBox

`CheckBox` *`starting x position, starting y position, width, height`*

For selecting one or more in a series of choices.

### Example

```
Sub Main()
   Begin Dialog DialogName1 60, 70, 160, 50, "ASC - Hello"
       CHECKBOX 42, 10, 48, 12, "", .checkInt
       OKBUTTON 42, 24, 40, 12
   End Dialog

   Dim Dlg1 As DialogName1
   Dialog Dlg1
   If Dlg1.checkInt = 0 Then
       Q = "didn't check the box."
     Else
       Q = "checked the box."
   End If
   MsgBox "You " & Q
End Sub
```

## Choose function

Choose( *number*, *choice1*, [*choice2*,] [*choice3*,]… )

Returns a value from a list of arguments.

Choose returns a null value if number is less than one or greater than the number of choices in the list. If number is not an integer it is rounded to the nearest integer.

### Example

```
Sub Main
   number = 2
   GetChoice = Choose(number, "Choice1", "Choice2", "Choice3")
   Print GetChoice
End Sub
```

## Chr function

Chr( *int* )

Returns a one-character string whose ASCII number is the argument.

Chr returns a String.

### Example

```
Sub ChrExample()
   Dim X, Y, Msg, NL
   NL = Chr(10)
   For X = 1 to 2
       For Y = Asc("A") To Asc("Z")
           Msg = Msg & Chr(Y)
           Next Y
   Msg = Msg & NL
   Next X
   MsgBox Msg
End Sub
```

## CInt function

CInt( *expression* )

Converts any valid expression to an integer.

### Example

```
Sub Main()
   Dim y As Long

   y = 25
   Print VarType(y)

   If  VarType(y) = 3 Then
       Print y
         x = CInt(y)    ' Converts the long value of y to an integer
                        ' value in x
       Print x
       Print VarType(x)
   End If
End Sub
```

## CLng function

CLng( *expression* )

Converts any valid expression into a long value.

### Example

```
Sub Main()
   Dim y As Integer

   y = 25000   ' The integer expression can only hold five digits.
   If  VarType(y) = 2 Then
       Print y
         x = CLng(y)    ' Converts the integer value of x to a long
                        ' value in x.
         x = x * 10000   ' y is now ten digits in the form of x.
       Print x
   End If
End Sub
```

## Close statement

Close [[*#filenumber*] [, [*#*]*filenumber*],,,

The Close statement takes one argument filenumber. Filenumber is the number used with the Open statement to open the file. If the Close statement is used without any arguments it closes all open files.

### Example

```
Sub Main
Open "c:\test.txt" For Input As #1
Do While Not EOF(1)
   MyStr = Input(10, #1)
   MsgBox MyStr
Loop
Close #1

End Sub

Sub Make3Files ()
   Dim I, FNum, FName   ' Declare variables.
   For I = 1 To 3
       FNum = FreeFile   ' Determine next file number.
       FName = "TEST" & FNum
       Open FName For Output As FNum   ' Open file.
       Print #I, "This is test #" & I   ' Write string to file.
       Print #I, "Here is another ";  "line";  I
   Next I
   Close   ' Close all files.
End Sub
```

## Const statement

`Const name = expression`

Assigns a symbolic name to a constant value.

A constant must be defined before it is used.

The definition of a `Const` in BlueZone Basic outside the procedure or at the module level is a global. The syntax Global Const and Const are used below outside the module level are identical.

A type declaration character can be used; however, if none is used BlueZone Basic automatically assigns one of the following data types to the constant: long (if it is a long or integer), Double (if a decimal place is present), or a String ( if it is a string).

### Example

```
Global Const Height = 14.4357
Const PI = 3.14159   'Global to all procedures in a module

Sub Main()
   Begin Dialog DialogName1 60, 60, 160,70, "ASC - Hello"
       TEXT 10, 10, 100, 20, "Please fill in the radius of circle x"
       TEXT 10, 40, 28, 12, "Radius"
       TEXTBOX 42, 40, 28, 12, .Radius
       OKBUTTON 42, 54,40, 12
   End Dialog

   Dim Dlg1 As DialogName1
   Dialog Dlg1
   CylArea = Height * (Dlg1.Radius * Dlg1.Radius) * PI
   MsgBox "The volume of Cylinder x is " & CylArea
End Sub
```

## Cos function

`Cos( rad )`

Returns the cosine of an angle.

The argument *rad* must be expressed in radians and must be a valid numeric expression. `Cos` by default returns a double unless a single or integer is specified as the return value.

### Example

```
Sub Main()
Dim J As Double
Dim I As Single   ' Declare variables.
Dim K As Integer
For I =1 To 10
   Msg = Msg & Cos(I) & ", "   ' Cos function call
   J=Cos(I)
   Print J
   K=Cos(I)
   Print K
Next I
MsgBox Msg   ' Display results.
MsgBox Msg1
End Sub
```

## CreateObject function

`CreateObject( class )`

Creates an OLE automation object.

### Example

```
Sub Command1_Click()
   Dim word6 As object
   Set word6 = CreateObject("Word.Basic")
   word6.FileNewDefault
   word6.InsertPara
   word6.Insert "Attn:"
   word6.InsertPara
   word6.InsertPara
   word6.InsertPara
   word6.Insert "     Vendor Name: "
   word6.Bold 1
   name = "Some Body"
   word6.Insert name
   word6.Bold 0
   word6.InsertPara
   word6.Insert "      Vendor Address:"
   word6.InsertPara
   word6.Insert "      Vendor Product:"
   word6.InsertPara
   word6.InsertPara
   word6.Insert "Dear Vendor:"
   word6.InsertPara
   word6.InsertPara
   word6.Insert "The letter you are reading was created with BlueZone Basic."
   word6.Insert "Using OLE Automation, BlueZone Basic can call any other OLE"
   word6.Insert " enabled application. BlueZone Basic is a Basic Scripting "
   word6.Insert "language for applications."
   word6.InsertPara
   word6.InsertPara
   word6.Insert "     Product Name:  BlueZone Basic"
   word6.InsertPara
   word6.Insert "     Company Name:  BlueZone Software"
   word6.InsertPara
   word6.InsertPara

        MsgBox "You have just called Word 6.0 using OLE"
End Sub
```

## CSng function

CSng( *expression* )

Converts any valid expression to a Single.

### Example

```
Sub Main()
   Dim y As Integer
   y = 25
   If  VarType(y) = 2 Then
      Print y
      x = CSng(y)   'Converts the integer value of y to a single value
                    ' in x.
      Print x
   End If
End Sub
```

# CStr function

CStr( *expression* )

Converts any valid expression to a String.

## Example

```
Sub Main
   Dim Y As Integer
     Y = 25
        Print Y
   If VarType(Y) = 2 Then
       X = CStr(Y)   ' Converts Y To a Str
       X = X + "hello"   ' It is now possible to combine Y with strings
        Print X
   End If
End Sub
```

# CurDir function

CurDir( *drive* )

Returns the current path for the specified drive.

CurDir returns a Variant; CurDir$ returns a String.

## Example

```
Declare Function CurDir Lib "NewFuns.dll" () As String

Sub Form_Click()
   Dim Msg, NL   ' Declare variables.
   NL = Chr(10)   ' Define newline.
   Msg = "The current directory is: "
   Msg = Msg & NL & CurDir()
   MsgBox Msg   ' Display message.
End Sub
```

# CVar function

CVar( *expression* )

Converts any valid expression to a Variant.

## Example

```
Sub Main
   Dim MyInt As Integer
   MyInt = 4534
   Print MyInt
   MyVar = CVar(MyInt & "0.23")   ' Makes MyInt a Variant + 0.32.
   Print MyVar
End Sub
```

# Date function

Date, Date()

Returns the current system date.

`Date` returns a Variant of `VarType 8 (String)` containing a date.

## Example

```
' Format Function Example
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main
   x = Date()
   Print Date
   Print x
   Print “VarType: “ & VarType(Date)
   MyTime = "08:04:23 PM"
   MyDate = "03/03/07"
   MyDate = "January 27, 2007"
   SysDate = Date
   MsgBox Sysdate,0,"System Date"

   MsgBox Now,0,"Now"
   MsgBox MyTime,0,"MyTime"

   MsgBox Second( MyTime ) & " Seconds"
   MsgBox Minute( MyTime ) & " Minutes"
   MsgBox Hour( MyTime ) & " Hours"

   MsgBox Day( MyDate ) & " Days"
   MsgBox Month( MyDate ) & " Months"
   MsgBox Year( MyDate ) & " Years"

   ' Returns current system time in the system-defined long time format.
   MsgBox Format(Time, "Short Time") & " Short Time"
   MsgBox Format(Time, "Long Time") & "Long Time"

   ' Returns current system date in the system-defined long date format.
   MsgBox Format(Date, "Short Date") & " Short Date"
   MsgBox Format(Date, "Long Date") & " Long Date"

   MyDate = "30 December 91"   ' Use of European date
   print Mydate

   MsgBox MyDate,0,"MyDate International..."
   MsgBox Day(MyDate),0,"day"
   MsgBox Month(MyDate),0,"month"
   MsgBox Year(MyDate),0,"year"

   MyDate = "30-Dec-91"   ' Another European date usage
   print Mydate

   MsgBox MyDate,0,"MyDate International..."
   MsgBox Day(MyDate),0,"day"
   MsgBox Month(MyDate),0," month"
   MsgBox Year(MyDate),0,"year"

   MsgBox Format("This is it", ">")   ' Returns "THIS IS IT".
End Sub
```

# DateSerial function

DateSerial( *year*, *month*,*day* )

Returns a variant (Date) corresponding to the year, month and day that were passed in. All three parameters for the DateSerial function are required and must be valid.

## Example

```
Sub Main
   Dim MDate
   MDate = DateSerial(1959, 5, 29)
   Print MDate
End Sub
```

# DateValue function

DateValue( *dateexpression* )

Returns a variant (Date) corresponding to the string date expression that was passed in. dateexpression can be a string or any expression that can represent a date, time or both a date and a time.

## Example

```
Sub Main()
Dim v As Variant
Dim d As Double
   d = Now
   Print d
   v = DateValue("1959/05/29")
   MsgBox (VarType(v))
   MsgBox (v)
End Sub
```

# Day function

Day( *dateexpression* )

Returns a variant date corresponding to the string date expression that was passed in. dateexpression can be a string or any expression that can represent a date.

## Example

```
Sub Main
    Dim MDate, MDay
    MDate = #May 29, 1959#
    MDay = Day(MDate)
    Print "The Day listed is the " & MDay
End Sub
```

# Declare statement

Declare Sub *procedurename Lib Libname$* [*Alias aliasname$*][( *argument list* )]

Declare Function *procedurename Lib Libname$* [*Alias aliasname$*] [( *argument list* )][*As Type*]

The `Declare` statement makes a reference to an external procedure in a Dynamic Link Library (DLL).

The `procedurename` parameter is the name of the function or subroutine being called.

The `Libname` parameter is the name of the DLL that contains the procedure.

The optional `Alias aliasname` clause, is used to supply the procedure name in the DLL if different from the name specified on the procedure parameter. When the optional argument list needs to be passed the format is as follows:

```
([ByVal] variable [As type] [,ByVal] variable [As type] ]…])
```

The optional `ByVal` parameter specifies that the variable is [passed by value instead of by reference (see “ByRef and ByVal” in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Variant.

Refer to Variable types, on page 100 for more information.

If a procedure has no arguments, use double parentheses () only to assure that no arguments are passed:

```
Declare Sub OntTime Lib “Check” ()
```

**Note**

BlueZone Basic extensions to the declare statement. The following syntax is not supported by Microsoft Visual Basic.

```
Declare Function procedurename App [Alias aliasname$] [(argument list)][As
Type]
```

This form of the `Declare` statement, makes a reference to a function located in the executable file located in the application where BlueZone Basic is embedded.

### Example

```
Declare Function GetFocus Lib "User" () As Integer
Declare Function GetWindowText Lib "User" (ByVal hWnd%, ByVal Mess$, ByVal cbMax%) _
As Integer

Sub Main
   Dim hWindow%
   Dim str1 As String *51
   Dim str2 As String * 25

   hWindow% = GetFocus()
   print "GetWindowText returned: ", GetWindowText( hWindow%, str1,51 )
   print "GetWindowText2 returned: ", GetWindowText( hWindow%, str2, 25)
   print str1
   print str2
End Sub
```

## Dialog function

```
Dialog( DialogRecord )
```

Returns a value corresponding to the button the user chooses.

The `Dialog()` function is used to display the dialog box specified by `DialogRecord`. `DialogRecord` is the name of the dialog and must be defined in a preceding `Dim` statement.

The return value or button:
-1 = OK button
0 = Cancel button

> 0 A command button where 1 is the first PushButton in the definition of the dialog and 2 is the second and so on.

## Example

```
' This sample shows all of the dialog controls on one dialog and how to
' vary the response based on which PushButton was pressed.

Sub Main()
   Dim MyList$(2)
   MyList(0) = "Banana"
   MyList(1) = "Orange"
   MyList(2) = "Apple"
   Begin Dialog DialogName1 60, 60, 240, 184, "Test Dialog"
       Text 10, 10, 28, 12, "Name:"
       TextBox 40, 10,50, 12, .joe
       ListBox 102, 10, 108, 16, MyList$(), .MyList1
       ComboBox 42, 30, 108, 42, MyList$(), .Combo1
       DropListBox 42, 76, 108, 36, MyList$(), .DropList1$
       OptionGroup .grp1
           OptionButton 42, 100, 48, 12, "Option&1"
           OptionButton 42, 110, 48, 12, "Option&2"
       OptionGroup .grp2
           OptionButton 42, 136, 48, 12, "Option&3"
           OptionButton 42, 146, 48, 12, "Option&4"
       GroupBox 132, 125, 70, 36, "Group"
       CheckBox 142, 100, 48, 12, "Check&A", .Check1
       CheckBox 142, 110, 48, 12, "Check&B", .Check2
       CheckBox 142, 136, 48, 12, "Check&C", .Check3
       CheckBox 142, 146, 48, 12, "Check&D", .Check4
       CancelButton 42, 168, 40, 12
       OKButton 90, 168, 40, 12
       PushButton 140, 168, 40, 12, "&Push Me 1"
       PushButton 190, 168, 40, 12, "Push &Me 2"
   End Dialog
   Dim Dlg1 As DialogName1
   Dim button As Integer
   Dlg1.joe  = "Def String"
   Dlg1.MyList1 = 1
   Dlg1.Combo1 = "Kiwi"
   Dlg1.DropList1 = 2
   Dlg1.grp2 = 1
   ' Dialog returns -1 for OK, 0 for Cancel, button # for PushButtons.
   button = Dialog( Dlg1 )
   'MsgBox "button: " & button   ' Uncomment for button return value.
   If button = 0 Then Exit Sub
   MsgBox "TextBox: "& Dlg1.joe
   MsgBox "ListBox: " & Dlg1.MyList1
   MsgBox Dlg1.Combo1
   MsgBox Dlg1.DropList1
   MsgBox "grp1: " & Dlg1.grp1
   MsgBox "grp2: " & Dlg1.grp2
   Begin Dialog DialogName2 60, 60, 160, 60, "Test Dialog 2"
       Text 10, 10, 28, 12, "Name:"
       TextBox 42, 10, 108, 12, .fred
       OkButton 42, 44, 40, 12
   End Dialog
   If button = 2 Then
       Dim Dlg2 As DialogName2
       Dialog Dlg2
       MsgBox Dlg2.fred
   ElseIf button = 1 Then
       Dialog Dlg1
       MsgBox Dlg1.Combo1
   End If
End Sub
```

# Dim statement

Dim *variablename*[( *subscripts* )][As *Type*][,name][As *Type*]]

Allocates storage for and declares the data type of variables and arrays in a module.

The types currently supported are integer, long, single, double and string and variant.

## Example

```
Sub Main
   Dim x As Long
   Dim y As Integer
   Dim z As single
   Dim a As double
   Dim s As String
   Dim v As Variant   ' This is the same as Dim x or Dim x As Any.
   Dim button As Integer
End Sub
```

# Dir function

Dir[( *path*,*attributes* )]

Returns a file/directory name that matches the given path and attributes.

### Example

```
'=======================================================
' Bitmap sample using the Dir Function
'=======================================================
Sub DrawBitmapSample
   Dim MyList()
   Begin Dialog BitmapDlg 60, 60, 290, 220, "BlueZone Basic bitmap sample", _ .DlgFunc
       ListBox 10, 10, 80, 180, MyList(), .List1, 2
       Picture 100, 10, 180, 180, "Forest.bmp", 0, .Picture1
       CancelButton 42, 198, 40, 12
       OKButton 90, 198, 40, 12
   End Dialog
   Dim frame As BitmapDlg
   ' Show the bitmap dialog
   Dialog frame
End Sub
Function DlgFunc( controlID As String, action As Integer, suppValue _ As Integer )
   DlgFunc = 1   ' Keep dialog active
   Select Case action
   Case 1   ' Initialize
       temp = Dir( "c:\Windows\*.bmp" )
       count = 0
       While temp <> ""
           count = count + 1
           temp = Dir
       Wend
       Dim x() As String
       ReDim x(count)
       x(0) = Dir( "c:\Windows\*.bmp" )
       For i = 1 To count
           x(i) = dir
       Next i
       DlgListBoxArray "List1", x()
   Case 2   ' Click
       fileName = "c:\windows\" & DlgText("List1")
       DlgSetPicture "Picture1", fileName
   End Select
End Function
```

## DlgEnable statement

DlgEnable "*ControlName*", *Value*

This statement is used to enable or disable a particular control on a dialog box.

The parameter ControlName is the name of the control on the dialog box.

The parameter Value is the value to set it to. 1 = Enable, 0 = Disable. On is equal to 1 in the example below. If the second parameter is omitted the status of the control toggles.

### Example

```
Begin Dialog UserDialog2 160,160, 260, 188, "3", .DlgFunc
   Text 8,10,73,13, "New dialog Label:"
   TextBox 8, 26, 160, 18, .FText
   CheckBox 8, 56, 203, 16, "New CheckBox",. ch1
   CheckBox 18,100,189,16, "Additional CheckBox", .ch2
   PushButton 18, 118, 159, 16, "Push Button", .but1
   OKButton 177, 8, 58, 21
   CancelButton 177, 32, 58, 21
End Dialog

Dim Dlg2 As UserDialog2
Dlg2.FText = "Your default string goes here"

Function DlgFunc( ControlID$, Action%, SuppValue% )

Select Case Action%

Case 1
   DlgEnable "Group", 0
   DlgVisible "Chk2", 0
   DlgVisible "History", 0
Case 2
   If ControlID$ = "Chk1" Then
       DlgEnable "Group", On
       DlgVisible "Chk2"
       DlgVisible "History"
   End If

   If ControlID$ = "Chk2" Then
       DlgText "History", "Push to display nested dialog"
   End If

   If ControlID$ = "History" Then
       DlgEnable=1
       Number = 4
       MsgBox SQR(Number) & " The sqr of 4 is 2"
       x = Dialog( Dlg2 )
   End If

   If ControlID$ = "but1" Then

   End If

Case Else

End Select
Enable =1

End Function
```

## DlgText statement

DlgText "*ControlName*", *String*

This statement is used to set or change the text of a dialog control.

The parameter `ControlName` is the name of the control on the dialog box. The parameter `String` is the value to set it to.

### Example

```
If ControlID$ = "Chk2" Then
   DlgText "History", "Push to display nested dialog"
End If
```

## DlgVisible statement

DlgVisible "*ControlName*", *Value*

This statement is used to hide or make visible a particular control on a dialog box.

The parameter ControlName is the name of the control on the dialog box. The parameter Value is the value to set it to. 1 = Visible, 0 = Hidden. On is equal to 1. If the second parameter is omitted the status of the control toggles.

### Example

```
If ControlID$ = "Chk1" Then
   DlgEnable "Group", On
   DlgVisible "Chk2"
   DlgVisible "History"
End If
```

## Do...Loop statement

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

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

Repeats a group of statements while a condition is true or until a condition is met.

### Example

```
Sub Main()
   Dim Value, Msg   ' Declare variables.
   Do
       Value = InputBox("Enter a value from 5 to 10.")
       If Value >= 5 And Value <= 10 Then
           Exit Do   ' Exit Do...Loop.
       Else
           Beep   ' Beep if not in range.
       End If
   Loop
End Sub
```

## End statement

End[{*Function* | *If* | *Sub*}]

Ends a program or a block of statements such as a Sub procedure or a function.

### Example

```
Sub Main()
   Dim Var1 as String

   Var1 = "hello"
   MsgBox " Calling Test"
   Test Var1
   MsgBox Var1

End Sub

Sub Test(wvar1 as string)

   wvar1 = "goodbye"
   MsgBox "Use of End Statement"
   End
End Sub
```

## EOF statement

EOF( *Filenumber* )

Returns a value during file input that indicates whether the end of a file has been reached.

### Example

```
' Input Function Example
' This example uses the Input function to read 10 characters at a time from a
' file and display them in a MsgBox.  This example assumes that TESTFILE is a
' text file with a few lines of 'sample data.

Sub Main
   Open "TESTFILE" For Input As #1    ' Open file.
   Do While Not EOF(1)    ' Loop until end of file.
       MyStr = Input(10, #1)    ' Get ten characters.
       MsgBox MyStr
   Loop
   Close #1    ' Close file.
End Sub
```

## Erase statement

Erase *arrayname*[,*arrayname*]

Re-initializes the elements of a fixed array.

### Example

```
' This example demonstrates some of the  features of arrays.  The lower bound
' for an array is 0 unless it is specified or option base has set it as is
' done in this example.

Option Base 1

Sub Main
' Declare array variables.
   Dim Num(10) As Integer   ' Integer array.
   Dim StrVarArray(10) As String   ' Variable-string array.
   Dim StrFixArray(10) As String * 10   ' Fixed-string array.
   Dim VarArray(10) As Variant   ' Variant array.
   Dim DynamicArray() As Integer   ' Dynamic array.
   ReDim DynamicArray(10)    ' Allocate storage space.
   Erase Num   ' Each element set to 0.
   Erase StrVarArray   ' Each element set to zero-length       string ("").
   Erase StrFixArray   ' Each element set to 0.
   Erase VarArray   ' Each element set to Empty.
   Erase DynamicArray   ' Free memory used by array.
End Sub
```

## Exit statement

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

Exits a loop or procedure.

### Example

```
' This sample shows Do...Loop with Exit Do to get out.

Sub Main()
   Dim Value, Msg   ' Declare variables.
   Do
       Value = InputBox("Enter a value from 5 to 10.")
       If Value >= 5 And Value <= 10 Then   ' Check range.
           Exit Do   ' Exit Do...Loop.
       Else
           Beep   ' Beep if not in range.
       End If
   Loop
End Sub
```

## Exp function

Exp( *num* )

Returns the base of the natural log raised to a power (e ^ num).

The value of the constant e is approximately 2.71828.

### Example

```
Sub ExpExample()
   ' Exp(x) is e ^x so Exp(1) is e ^1 or e.

   Dim Msg, ValueOfE    ' Declare variables.
   ValueOfE = Exp(1)   ' Calculate value of e.
   Msg = "The value of e is " & ValueOfE
   MsgBox Msg    ' Display message.
End Sub
```

## FileCopy function

`FileCopy( sourcefile, destinationfile )`

Copies a file from source to destination.

The `sourcefile` and `destinationfile` parameters must be valid string expressions. `sourcefile` is the file name of the file to copy, `destinationfile` is the file name to be copied to.

### Example

```
Dim SourceFile, DestinationFile
SourceFile = "SRCFILE"    ' Define source file name.
DestinationFile = "DESTFILE"    ' Define target file name.
FileCopy SourceFile, DestinationFile    ' Copy source to target.
```

## FileLen function

`FileLen( filename )`

Returns a Long integer that is the length of the file in bytes.

### Example

```
Sub Main
   Dim MySize
   MySize = FileLen("C:\TESTFILE")    ' Returns file length (bytes).
   Print MySize
End Sub
```

## Fix function

`Fix( number )`

Returns the integer portion of a number.

### Example

```
Sub Main
   Dim MySize
   MySize = Fix(4.345)
   Print MySize
End Sub
```

## For Each...Next statement

`For  Each element in group`

```
    [statements]
    [Exit For]
    [statements]
Next    [element]
```

Repeats the group of statements for each element in an array of a collection. `For each ... Next` statements can be nested if each loop element is unique. The `For Each...Next` statement cannot be used with and array of user defined types.

### Example

```
Sub Main
   dim z(1 to 4) as double
   z(1) = 1.11
   z(2) = 2.22
   z(3) = 3.33
   For Each v In z
       Print v
   Next v
End Sub
```

## For...Next statement

```
For  counter = expression1 to expression2 [Step increment]
     [statements]
Next     [counter]
```

Repeats the execution of a block of statements for a specified number of times.

### Example

```
Sub main ()
Dim x,y,z

   For x = 1 to 5
       For y = 1 to 5
           For z = 1 to 5
           Print "Looping" ,z,y,x
           Next z
       Next y
   Next x
End Sub
```

## Format function

```
Format( expression [,fmt ] )
```

Formats a string, number or variant data type to a format expression.

`Format` returns a string.

**`expression`**
Expression to be formatted.

**`fmt`**
A string of characters that specify how the expression is to displayed. or the name of a commonly-used format that has been predefined in BlueZone Basic. Do not mix different type format expressions in a single fmt parameter.

If the `fmt` parameter is omitted or is zero-length and the expression parameter is a numeric, `Format[$]` provides the same functionality as the `Str[$]` function by converting the numeric value to the appropriate return data type, Positive numbers convert to strings using `Format[$]` lack the leading space reserved for displaying the sign of the value, whereas those converted using `Str[$]` retain the leading space.

To format numbers, you can use the commonly-used formats that have been predefined in BlueZone Basic, or you can create user-defined formats with standard characters that have special meaning when used in a format expression.

Table 13: Predefined numeric formats lists the predefined numeric format names.

Table 13: Predefined numeric formats

| Format name | Description |
|---|---|
| General Number | Display the number as is, with no thousand Separators. |
| Fixed | Display at least one digit to the left and two digits to the right of the decimal separator. |
| Standard | Display number with thousand separator, if appropriate; display two digits to the right of the decimal separator. |
| Percent | Display number multiplied by 100 with a percent sign (%) appended to the right' display two digits to the right of the decimal separator. |
| Scientific | Use standard scientific notation. |
| True/False | Display False if number is 0, otherwise display True. |

Table 14: User-defined characters shows the characters available to create user-defined number formats.

Table 14: User-defined characters

| Character | Meaning |
|---|---|
| Null string | Display the number with no formatting. |
| 0 | Digit placeholder. |

**Display a digit or a zero**

If the number being formatted has fewer digits than there are zeros (on either side of the decimal) in the format expression, leading or trailing zeros are displayed. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, the number is rounded to as many decimal places as there are zeros. If the number has more digits to left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, the extra digits are displayed without modification.

**# Digit placeholder**

Displays a digit or nothing. If there is a digit in the expression being formatted in the position where the # appears in the format string, displays it; otherwise, nothing is displayed.

**. Decimal placeholder**

The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator.

**% Percentage placeholder**

The percent character (%) is inserted in the position where it appears in the format string. The expression is multiplied by 100.

**, Thousand separator**

The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator.

Use of this separator as specified in the format statement contains a comma surrounded by digit placeholders (0 or #). Two adjacent commas or a comma immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed."

**E-E+e-e+ Scientific format**

If the format expression contains at least one digit placeholder (0 or #) to the right of E-,E+,e- or e+, the number is displayed in scientific formatted E or e inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a plus sign next to positive exponents.

**: Time separator**

The actual character used as the time separator depends on the Time Format specified in the International section of the Control Panel.

**/ Date separator**

The actual character used as the date separator in the formatted out depends on Date Format specified in the International section of the Control Panel.

**- + $ ( ) space Display a literal character**

To display a character other than one of those listed, precede it with a backslash (\).

**\ Display the next character in the format string**

The backslash itself isn't displayed. To display a backslash, use two backslashes (\\).

Examples of characters that can't be displayed as literal characters are the date- and time-formatting characters (a,c,d,h,m,n,p,q,s,t,w,y, and /:), the numeric -formatting characters (#,0,%,E,e,comma, and period), and the string- formatting characters (@,&,<,>, and !).

**"String" Display the string inside the double quotation marks**

To include a string in fmt from within BlueZone Basic, you must use the ANSI code for a double quotation mark Chr(34) to enclose the text.

*** Display the next character as the fill character**

Any empty space in a field is filled with the character following the asterisk.

Unless the `fmt` argument contains one of the predefined formats, a format expression for numbers can have from one to four sections separated by semicolons.

| If you use | The result is |
|---|---|
| One section only | The format expression applies to all values. |
| Two | The first section applies to positive values, the second to negative sections values. |
| Three | The first section applies to positive values, the second to negative sections values, and the third to zeros. |
| Four | The first section applies to positive values, the second to negative section values, the third to zeros, and the fourth to Null values. |

The following example has two sections: the first defines the format for positive values and zeros; the second section defines the format for negative values.

```
"$#,##0; ($#,##0)"
```

If you include semicolons with nothing between them. the missing section is printed using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero.

"$#,##0;;\Z\e\r\o"

The following table shows sample format expressions for numbers. These examples all assume the Country is set to United States in the International section of the Control Panel. The first column contains the format strings. The other columns contain the output the results if the formatted data has the value given in the column headings.

Table 15: Sample format expressions

| Format (fmt) | Positive 3 | Negative 3 | Decimal .3 | Null |
|---|---|---|---|---|
| Null string | 3 | -3 | 0.3 | |
| 0 | 3 | -3 | 1 | |
| 0.00 | 3.0 | -3.00 | 0.30 | |
| #,##0 | 3 | -3 | 1 | |
| #,##0.00;;;Nil | 3.0 | -3.00 | 0.30 | Nil |
| $#,##0;($#,##0) | $3 | ($3) | $1 | |
| $#,##0.00;($#,##0.00) | $3.0 | ($3.00) | $0.30 | |
| 0% | 300% | -300% | 30% | |
| 0.00% | 300.00% | -300.00% | 30.00% | |
| 0.00E+00 | 3.00E+00 | -3.00E+00 | 3.00E-01 | |
| 0.00E-00 | 3.00E00 | -3.00E00 | 3.00E-01 | |

Numbers can also be used to represent date and time information. You can format date and time serial numbers using date and time formats or number formats because date/time serial numbers are stored as floating-point values.

To format dates and times, you can use either the commonly used format that have been predefined or create user-defined time formats using standard meaning of each:

The following table shows the predefined data format names you can use and the meaning of each.

Table 16: Data format names

| Format name | Description |
|---|---|
| General | Display a date and/or time. For real numbers, display a date and time (for example, 4/3/93 03:34 p.m.); if there is no fractional part, display only a date (for example, 4/3/93); if there is no integer part, display time only (for example, 03:34 p.m.). |
| Long Date | Display a Long Date, as defined in the International section of the Control Panel. |
| Medium | Display a date in the same form as the Short Date, as defined in the international section of the Control Panel, except spell out the month abbreviation. |
| Short Date | Display a Short Date, as defined in the International section of the Control Panel. |
| Long Time | Display a Long Time, as defined in the International section of the Control panel. Long Time includes hours, minutes, seconds. |
| Medium Time | Display time in 12-hour format using hours and minuets and the Time AM/PM designator. |
| Short Time | Display a time using the 24-hour format (for example, 17:45) |

The following table shows the characters you can use to create user-defined date/time formats.

Table 17: User-defined date/time format characters

| Character | Meaning |
|---|---|
| c | Display the date as dddd and display the time as ttttt. in the order. |
| d | Display the day as a number without a leading zero (1-31). |
| dd | Display the day as a number with a leading zero (01-31). |
| ddd | Display the day as an abbreviation (Sun-Sat). |
| ddddd | Display a date serial number as a complete date (including day , month, and year). |
| w | Display the day of the week as a number (1- 7 ). |
| ww | Display the week of the year as a number (1-53). |
| m | Display the month as a number without a leading zero (1-12). If m immediately follows h or hh, the minute rather than the month is displayed. |
| mm | Display the month as a number with a leading zero (01-12). If mm immediately follows h or hh, the minute rather than the month is displayed. |
| mmm | Display the month as an abbreviation (Jan-Dec). |
| mmmm | Display the month as a full month name (January-December). |
| q | Display the quarter of the year as a number (1-4). |
| y | Display the day of the year as a number (1-366). |
| yy | Display the day of the year as a two-digit number (00-99) |
| yyyy | Display the day of the year as a four-digit number (100-9999). |
| h | Display the hour as a number without leading zeros (0-23). |
| hh | Display the hour as a number with leading zeros (00-23). |
| n | Display the minute as a number without leading zeros (0-59). |
| nn | Display the minute as a number with leading zeros (00-59). |
| s | Display the second as a number without leading zeros (0-59). |
| ss | Display the second as a number with leading zeros (00-59). |
| ttttt | Display a time serial number as a complete time (including hour, minute, and second) formatted using the time separator defined by the Time Format in the International section of the Control Panel. A leading zero is displayed if the Leading Zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is h:mm:ss. |
| AM/PM | Use the 12-hour clock and display an uppercase AM/PM |
| am/pm | Use the 12-hour clock display a lowercase am/pm |
| A/P | Use the 12-hour clock display a uppercase A/P |
| a/p | Use the 12-hour clock display a lowercase a/p |
| AMPM | Use the 12-hour clock and display the contents of the 11:59 string (s1159) in the WIN.INI file with any hour before noon; display the contents of the 2359 string (s2359) with any hour between noon and 11:59 PM. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as it exists in the WIN.INI file. The default format is AM/PM. |

The following table shows examples of user-defined date and time formats:

Table 18: User-defined date/time formats

| Format | Display |
|---|---|
| m/d/yy | 2/26/65 |
| d-mmmm-yy | 26-February-65 |
| d-mmmm | 26 February |
| mmmm-yy | February 65 |
| hh:nn | AM/PM 06:45 PM |
| h:nn:ss a/p | 6:45:15 p |
| h:nn:ss | 18:45:15 |
| m/d/yy/h:nn | 2/26/65 18:45 |

Strings can also be formatted with Format[$]. A format expression for strings can have one section or two sections separated by a semicolon.

Table 19: Format expressions for strings sections

| If you use | The result is |
|---|---|
| One section only | The format applies to all string data. |
| Two sections | The first section applies to string data, the second to Null values and zero-length strings. |

The following characters can be used to create a format expression for strings:

Table 20: Characters for string format expressions

| Character | Meaning |
|---|---|
| @ | Character placeholder. |
| | Displays a character or a space. Placeholders are filled from right to left unless there is an ! character in the format string. |
| & | Character placeholder. Display a character or nothing. |
| < | Force lowercase. |
| > | Force uppercase. |
| ! | Force placeholders to fill from left to right instead of right to left. |

## Example

```
' Format Function Example
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.
' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main
   MyTime = "08:04:23 PM"
   MyDate = "03/03/07"
   MyDate = "January 27, 2007"
   MsgBox Now
   MsgBox MyTime
   MsgBox Second( MyTime ) & " Seconds"
   MsgBox Minute( MyTime ) & " Minutes"
   MsgBox Hour( MyTime ) & " Hours"
   MsgBox Day( MyDate ) & " Days"
   MsgBox Month( MyDate ) & " Months"
   MsgBox Year( MyDate ) & " Years"

   ' Returns current system time in the system-defined long time format.
   MsgBox Format(Time, "Short Time")
   MyStr = Format(Time, "Long Time")

   ' Returns current system date in the system-defined long date format.
   MsgBox Format(Date, "Short Date")
   MsgBox Format(Date, "Long Date")
   MyStr Format(MyTime, "h:n:s")   ' Returns "17:4:23".
   MyStr Format(MyTime, "hh:nn:ss")   ' Returns "20:04:22 ".
   MyStr Format(MyDate, "dddd, mmm d yyyy")   ' Returns "Wednesday,
                                              ' Jan 27 2007".

   ' If format is not supplied, a string is returned.
   MsgBox Format(23)   ' Returns "23".

   ' User-defined formats.
   MsgBox Format(5459.4, "##,##0.00")   ' Returns "5,459.40".
   MsgBox Format(334.9, "###0.00")   ' Returns "334.90".
   MsgBox Format(5, "0.00%")   ' Returns "500.00%".
   MsgBox Format("HELLO", "<")   ' Returns "hello".
   MsgBox Format("This is it", ">")   ' Returns "THIS IS IT".
End Sub
```

# FreeFile function

`FreeFile`

Returns an integer that is the next available file handle to be used by the `Open` statement.

### Example

```
Sub Main
Dim Mx, FileNumber

   For Mx = 1 To 3
       FileNumber = FreeFile
       Open "c:\e1\TEST" & Mx For Output As #FileNumber
       Write #FileNumber, "This is a sample."
       Close #FileNumber
   Next Mx

   Open "c:\e1\test1" For Input As #1
   Do While Not EOF(1)
       MyStr = Input(10, #1)
       MsgBox MyStr
   Loop
   Close #1
End Sub
```

## Function statement

```
Function Fname [(Arguments)] [As type]
[statements]
Functionname = expression
[statements]
Functionname = expression
End Function
```

Declares and defines a procedure that can receive arguments and return a value of a specified data type.

When the optional argument list needs to be passed the format is as follows:

```
([ByVal] variable [As type] [,ByVal] variable [As type] ]…])
```

The optional `ByVal` parameter specifies that the variable is [passed by value instead of by reference (see "ByRef and ByVal" in this manual). The optional As type parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Variant.

Refer to Variable types, on page 100 for additional information.

### Example

```
Sub Main
Dim I as integer
   For I = 1 to 10
   Print GetColor2(I)
   Next I
End Sub

Function GetColor2( c% ) As Long
   GetColor2 = c% * 25
   If c% > 2 Then
       GetColor2 = 255   ' 0x0000FF - Red
   End If

   If c% > 5 Then
       GetColor2 = 65280   ' 0x00FF00 - Green
   End If

   If c% > 8 Then
       GetColor2 = 16711680   ' 0xFF0000 - Blue
   End If
End Function
```

## Get statement

`GetStatement [#] filenmber,[recordnumber], variablename`

Reads from a disk file into a variable. The `Get` statement has these parts:

**`Filenumber`**
The number used to Open the file with.

**`Recordnumber`**
For files opened in Binary mode recordnumber is the byte position where reading starts.

**`VariableName`**
The name of the variable used to receive the data from the file.

## GetObject function

`GetObject( filename[,class] )`

The `GetObject` function has two parameters:

**`filename`**
The name of the file containing the object to retrieve. If `filename` is an empty string then class is required.

**`class`**
A string containing the class of the object to retrieve.

## Global statement

`Global Const constant`

The `Global` statement must be outside the procedure section of the script. `Global` variables are available to all functions and subroutines in your program.

### Example

```
Global Const Height = 14.4357
Const PI = 3.14159   ' Global to all procedures in a module.

Sub Main()
   Begin Dialog DialogName1 60, 60, 160,70, "ASC - Hello"
       TEXT 10, 10, 100, 20, "Please fill in the radius of circle x"
       TEXT 10, 40, 28, 12, "Radius"
       TEXTBOX 42, 40, 28, 12, .Radius
       OKBUTTON 42, 54,40, 12
   End Dialog

Dim Dlg1 As DialogName1
Dialog Dlg1
CylArea = Height * (Dlg1.Radius * Dlg1.Radius) * PI
MsgBox "The volume of Cylinder x is " & CylArea
End Sub
```

## GoTo statement

GoTo *label*

Branches unconditionally and without return to a specified `label` in a procedure.

### Example

```
Sub Main()
   Dim x,y,z

   For x = 1 to 5
       For y = 1 to 5
           For z = 1 to 5
           Print "Looping" ,z,y,x
               If y > 3 Then
                   GoTo Label1
               End If
           Next z
       Next y
   Next x
Label1:
End Sub
```

## Hex function

Hex( *num* )

Returns the hexadecimal value of a decimal parameter. `Hex` returns a string.

The parameter `num` can be any valid number. It is rounded to nearest whole number before evaluation.

### Example

```
Sub Main()
   Dim Msg As String, x%
   x% = 10
   Msg =Str( x%) &  " decimal is "
   Msg = Msg & Hex(x%) & " in hex "
   MsgBox Msg
End Sub
```

# Hour function

Hour( *string* )

The Hour function returns an integer between 0 and 23 that is the hour of the day indicated in the parameter number.

The parameter string is any number expressed as a string that can represent a date and time from January 1, 1980 through December 31, 9999.

### Example

```
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.

' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main
   MyTime = "08:04:23 PM"
   MyDate = "03/03/07"
   MyDate = "January 27, 2007"

   MsgBox Now
   MsgBox MyTime

   MsgBox Second( MyTime ) & " Seconds"
   MsgBox Minute( MyTime ) & " Minutes"
   MsgBox Hour( MyTime ) & " Hours"

   MsgBox Day( MyDate ) & " Days"
   MsgBox Month( MyDate ) & " Months"
   MsgBox Year( MyDate ) & " Years"

   ' Returns current system time in the system-defined long time format.
   MsgBox Format(Time, "Short Time")
   MyStr = Format(Time, "Long Time")

   ' Returns current system date in the system-defined long date format.
   MsgBox Format(Date, "Short Date")
   MsgBox Format(Date, "Long Date")

   ' This section not yet supported
   'MyStr = Format(MyTime, "h:n:s")   ' Returns "17:4:23".
   'MyStr = Format(MyTime, "hh:nn:ss AMPM")   ' Returns "05:04:23 PM".
   'MyStr = Format(MyDate, "dddd, nnn d yyyy")   ' Returns "Wednesday, _
                                                ' Jan 27 2007".

   ' If format is not supplied, a string is returned.
   MsgBox Format(23)   ' Returns "23".

   ' User-defined formats.
   MsgBox Format(5459.4, "##,##0.00")   ' Returns "5,459.40".
   MsgBox Format(334.9, "###0.00")   ' Returns "334.90".
   MsgBox Format(5, "0.00%")   ' Returns "500.00%".
   MsgBox Format("HELLO", "<")   ' Returns "hello".
   MsgBox Format("This is it", ">")   ' Returns "THIS IS IT".
End Sub
```

## HTMLDialog

HTMLDialog( *path*, *number* )

Runs a DHTML dialog that is specified in the path.

### Example

```
x =HtmlDialog( "c:\htmlt.htm", 57 )
```

# If...Then...Else statement

### Syntax 1

```
If condition Then thenpart [Else elsepart]
```

### Syntax 2

```
If condition Then

   [statement(s)]

ElseIf condition Then

   [statement(s)]

Else

   [statements(s)]

End If

Syntax 2
If conditional Then statement
```

Allows conditional statements to be executed in the code.

### Example

```
Sub IfTest
   ' demo If...Then...Else
   Dim msg as String
   Dim nl as String
   Dim someInt as Integer

   nl = Chr(10)
   msg = "Less"
   someInt = 4

   If 5 > someInt Then msg = "Greater" : Beep
   MsgBox "" & msg

   If 3 > someInt Then
       msg = "Greater"
       Beep
   Else
       msg = "Less"
   End If
   MsgBox "" & msg

   If someInt = 1 Then
       msg = "Spring"
   ElseIf someInt = 2 Then
       msg = "Summer"
   ElseIf someInt = 3 Then
       msg = "Fall"1
   ElseIf someInt = 4 Then
       msg = "Winter"
   Else
       msg = "Salt"
   End If
   MsgBox "" & msg
End Sub
```

## Input # statement

`Input # filenumber, variablelist`

`Input #` statement reads data from a sequential file and assigns that data to variables.

The `Input #` statement has two parameters:

**`filenumber`**
The number used in the open statement when the file was opened.

**`variablelist`**
A comma-delimited list of the variables that are assigned when read from the file.

### Example

```
Dim MyString, MyNumber
Open "c:\TESTFILE" For Input As #1    ' Open file for input.
Do While Not EOF(1)    ' Loop until end of file.
   Input #1, MyString, MyNumber    ' Read data into two variables.
Loop
Close #1    ' Close file.
```

# Input function

Input( *n* , [#] *filenumber* )

Input returns characters from a sequential file.

The Input function has two parameters:

**n**
The number of bytes to be read from a file.

**filenumber**
The number used in the open statement when the file was opened.

## Example

```
Sub Main
   Open "TESTFILE" For Input As #1   ' Open file.
   Do While Not EOF(1)   ' Loop until end of file.
       MyStr = Input(10, #1)   ' Get ten characters.
       MsgBox MyStr
   Loop
   Close #1   ' Close file.
End Sub
```

# InputBox function

InputBox( *prompt*[,[*title*][,[*default*][,*xpos*,*ypos*]]] )

InputBox returns a String.

**prompt**
A string that is displayed usually to ask for input type or information.

**title**
A string that is displayed at the top of the input dialog box.

**default**
A string that is displayed in the text box as the default entry.

**xpos and ypos**
The x and y coordinates of the relative location of the input dialog box.

## Example

```
Sub Main
   Title$ = "Greetings"
   Prompt$ = "What is your name?"
   Default$ = ""
   X% = 200
   Y% = 200
   N$ = InputBox$(Prompt$, Title$, Default$, X%, Y%)
End Sub
```

# InStr function

InStr( *numbegin*, *string1*, *string2* )

Returns the character position of the first occurrence of string2 within string1.

**numbegin**
Required parameter. Sets the starting point of the search. Must be a valid positive integer no greater than 65,535.

**string1**
The string being searched.

**string2**
The string you are looking for.

### Example

```
Sub Main
   B$ = "Good Bye"
   A% = InStr(2, B$, "Bye")
   C% = Instr(3, B$, "Bye")
End Sub
```

## Int function

`Int( number )`

Returns the integer portion of a number.

## IsArray function

`IsArray( variablename )`

Returns a boolean value True or False, indicating whether the parameter `vaiablename` is an array.

### Example

```
Sub Main
   Dim MArray(1 To 5) As Integer, MCheck
   MCheck = IsArray(MArray)
   Print MCheck
End Sub
```

## IsDate function

`IsDate( variant )`

Returns a value that indicates if a variant parameter can be converted to a date.

### Example

```
Sub Main
Dim x As String
Dim MArray As Integer, MCheck
   MArray = 345
   x = "January 1, 1959"
   MCheck = IsDate(MArray)
   MChekk = IsDate(x)
   MArray1 = CStr(MArray)
   MCheck1 = CStr(MCheck)
   Print MArray1 & " is a date " & Chr(10) & MCheck
   Print x & " is a date" & Chr(10) & MChekk
End Sub
```

# IsEmpty function

`IsEmpty( variant )`

Returns a value that indicates if a variant parameter has been initialized.

## Example

```
' This sample explores the concept of an empty variant.

Sub Main
   Dim x   ' Empty
   x = 5   ' Not Empty - Long
   x = Empty   ' Empty
   y = x   ' Both Empty
   MsgBox “x” & ” IsEmpty: ” & IsEmpty(x)
End Sub
```

# IsNull function

`IsNull( v )`

Returns a value that indicates if a variant contains the NULL value.

The parameter v can be any variant. `IsNull` returns a TRUE if v contains NULL. If `IsNull` returns a FALSE the variant expression is not NULL.

The NULL value is special because it indicates that the v parameter contains no data. This is different from a null-string, which is a zero length string and an empty string which has not yet been initialized.

# IsNumeric function

`IsNumeric( v )`

Returns a TRUE or FALSE indicating if the v parameter can be converted to a numeric data type.

The parameter v can be any variant, numeric value, Date or string (if the string can be interpreted as a numeric).

## Example

```
Sub Form_Click()
   Dim TestVar   ' Declare variable.
   TestVar = InputBox("Please enter a number, letter, or symbol.")
   If IsNumeric(TestVar) Then   ' Evaluate variable.
       MsgBox "Entered data is numeric."   ' Message if number.
   Else
       MsgBox "Entered data is not numeric."   ' Message if not.
   End If
End Sub
```

# IsObject function

`IsObject( objectname )`

Returns a boolean value True or False indicating whether the parameter `objectname` is an object.

### Example

```
Sub Main
   Dim MyInt As Integer, MyCheck
   Dim MyObject As Object
   Dim YourObject As Object
   Set MyObject = CreateObject("Word.Basic")

   Set YourObject = MyObject
   MyCheck = IsObject(YourObject)

       Print MyCheck
End Sub
```

## Kill statement

`Kill` *filename*

Kill only deletes files. To remove a directory use the `RmDir` statement. Refer to RmDir statement, on page 183 for more information.

### Example

```
Const NumberOfFiles = 3

Sub Main()
   Dim Msg   ' Declare variable.
   Call MakeFiles()   ' Create data files.
   Msg = "Several test files have been created on your disk. You may see "
   Msg = Msg & "them by switching tasks. Choose OK to remove the _
   test files."
   MsgBox Msg
   For I = 1 To NumberOfFiles
       Kill "TEST" & I   ' Remove data files from disk.
   Next I
End Sub

Sub MakeFiles()
   Dim I, FNum, FName   ' Declare variables.
   For I = 1 To NumberOfFiles
       FNum = FreeFile   ' Determine next file number.
       FName = "TEST" & I
       Open FName For Output As FNum   ' Open file.
       Print #FNum, "This is test #" & I   ' Write string to file.
       Print #FNum, "Here is another ";  "line";  I
   Next I
   Close   ' Close all files.
End Sub
```

## LBound function

`LBound(` *array* `[,`*dimension*`] )`

Returns the smallest available subscript for the dimension of the indicated array.

### Example

```
' This example demonstrates some of the features of arrays.  The
' lower bound for an array is 0 unless it is specified or option
' base has set as is done in this example.

Option Base 1

Sub Main
   Dim a(10) As Double
   MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
   Dim i As Integer
   For i = 0 to 3
       a(i) = 2 + i * 3.1
   Next i
   Print a(0),a(1),a(2), a(3)
End Sub
```

## LCase function

Lcase[$]( *string* )

Returns a `string` in which all letters of the string parameter have been converted to lowercase.

### Example

```
' This example uses the LTrim and RTrim functions to strip leading
' and trailing spaces, respectively, from a string variable.  It
' uses the Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls.

Sub Main
   MyString = "  <-Trim->  "   ' Initialize string.
   TrimString = LTrim(MyString)   ' TrimString = "<-Trim->  ".
   MsgBox "|" & TrimString & "|"
   TrimString = LCase(RTrim(MyString))   ' TrimString = "  <-trim->".
   MsgBox "|" & TrimString & "|"
   TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->".
   MsgBox "|" & TrimString & "|"

   ' Using the Trim function alone achieves the same result.
   TrimString = UCase(Trim(MyString))   ' TrimString = "<-TRIM->".
   MsgBox "|" & TrimString & "|"
End Sub
```

## Left function

Left( *string*, *num* )

Returns the left most `num` characters of a `string` parameter.

`Left` returns a Variant, `Left$` returns a String.

### Example

```
Sub Main()
   Dim LWord, Msg, RWord, SpcPos, UsrInp   ' Declare variables.
   Msg = "Enter two words separated by a space."
   UsrInp = InputBox(Msg)   ' Get user input.
   print UsrInp
   SpcPos = InStr(1, UsrInp, " ")   ' Find space.
   If SpcPos Then
       LWord = Left(UsrInp, SpcPos - 1)   ' Get left word.
       print "LWord: ";  LWord
       RWord = Right(UsrInp, Len(UsrInp) - SpcPos)   ' Get right word.
       Msg = "The first word you entered is " & LWord
       Msg = Msg & "." & " The second word is "
       Msg = "The first word you entered is <" & LWord & ">"
       Msg = Msg & RWord & "."
   Else
       Msg = "You didn't enter two words."
   End If
   MsgBox Msg   ' Display message.
   MidTest = Mid("Mid Word Test", 4, 5)
   Print MidTest
End Sub
```

## Len function

Len( *string* )

Returns the number of characters in a string.

### Example

```
Sub Main
   A$ = "BlueZone Basic"
   StrLen% = Len(A$)   'the value of StrLen is 14
   MsgBox StrLen%
End Sub
```

## Let statement

[Let] *variablename* = *expression*

Let assigns a value to a variable.

Let is an optional keyword that is rarely used. The Let statement is required in older versions of BASIC.

### Example

```
Sub Form_Click()
   Dim Msg, Pi   ' Declare variables.
   Let Pi = 4 * Atn(1)   ' Calculate Pi.
   Msg = "Pi is equal to " & Str(Pi)
   MsgBox Msg   ' Display results.
End Sub
```

## Line Input # statement

Line Input # *filenumber* and *name*

Reads a line from a sequential file into a String or Variant variable.

**filenumber**
Used in the open statement to open the file.

**name**
The name of a variable used to hold the line of text from the file.

### Example

```
' Line Input # Statement Example:
' This example uses the Line Input # statement to read a line from a
' sequential file and assign it to a variable. This example assumes that
' TESTFILE is a text file with a few lines of sample data.

Sub Main
   Open "TESTFILE" For Input As #1    ' Open file.
   Do While Not EOF(1)    ' Loop until end of file.
       Line Input #1, TextLine    ' Read line into variable.
       Print TextLine    ' Print to Debug window.
   Loop
   Close #1    ' Close file.
End Sub
```

## LOF function

LOF( *filenumber* )

Returns a long number for the number of bytes in the open file.

The parameter `filenumber` is required and must be an integer.

### Example

```
Sub Main
   Dim FileLength
   Open "TESTFILE" For Input As #1
   FileLength = LOF(1)
   Print FileLength
   Close #1
End Sub
```

## Log function

Log( *num* )

Returns the natural log of a number.

The parameter `num` must be greater than zero and be a valid number.

### Example

```
Sub Form_Click( )
   Dim I, Msg, NL
   NL = Chr(13) & Chr(10)
   Msg = Exp(1) & NL
   For I = 1 to 3
   Msg = Msg & Log(Exp(1) ^ I ) & NL
   Next I
   MsgBox Msg
End Sub
```

# Mid function

*string* = Mid( *strgvar*,*begin*,*length* )

Returns a substring within a string.

## Example

```
Sub Main()

Dim LWord, Msg, RWord, SpcPos, UsrInp   ' Declare variables.

   Msg = "Enter two words separated by a space."
   UsrInp = InputBox(Msg)   ' Get user input.
   print UsrInp
   SpcPos = InStr(1, UsrInp, " ")   ' Find space.
   If SpcPos Then
       LWord = Left(UsrInp, SpcPos - 1)   ' Get left word.
       print "LWord: ";  LWord
       RWord = Right(UsrInp, Len(UsrInp) - SpcPos)   ' Get right word.
       Msg = "The first word you entered is " & LWord
       Msg = Msg & "." & " The second word is "
       Msg = "The first word you entered is <" & LWord & ">"
       Msg = Msg & RWord & "."
   Else
       Msg = "You didn't enter two words."
   End If
   MsgBox Msg   ' Display message.
   MidTest = Mid("Mid Word Test", 4, 5)
   Print MidTest
End Sub
```

# Minute function

Minute( *string* )

Returns an integer between 0 and 59 representing the minute of the hour.

### Example

```
' Format Function Example

' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.
' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main
   MyTime = "08:04:23 PM"
   MyDate = "03/03/07"
   MyDate = "January 27, 2007"

   MsgBox Now
   MsgBox MyTime

   MsgBox Second( MyTime ) & " Seconds"
   MsgBox Minute( MyTime ) & " Minutes"
   MsgBox Hour( MyTime ) & " Hours"
   MsgBox Day( MyDate ) & " Days"
   MsgBox Month( MyDate ) & " Months"
   MsgBox Year( MyDate ) & " Years"
End Sub
```

## MkDir function

MkDir *path*

Creates a new directory.

The parameter path is a string expression that must contain fewer than 128 characters.

### Example

```
Sub Main
   Dim DST As String
   DST = "t1"
   mkdir DST
   mkdir "t2"
End Sub
```

## Month function

Month( *number* )

Returns an integer between 1 and 12, inclusive, that represents the month of the year.

### Example

```
Sub Main
   MyDate = "03/03/07"
   print MyDate
    x = Month(MyDate)
   print x
End Sub
```

## MsgBox function

MsgBox( *msg*, [*type*] [, *title*] )

Displays a message in a dialog box and waits for the user to choose a button.

**msg**
The string displayed in the dialog box as the message.

**type**
Optional parameter. Designates the type of button.

**title**
Optional parameter. Designates the dialog box title. If you omit the argument title, MsgBox has no default title.

The MsgBox function returns a value indicating which button the user has selected. The MsgBox statement does not.

| Value | Meaning |
|---|---|
| 0 | Display OK button only. |
| 1 | Display OK and Cancel buttons. |
| 2 | Display Abort, Retry, and Ignore buttons. |
| 3 | Display Yes, No, and Cancel buttons. |
| 4 | Display Yes and No buttons. |
| 5 | Display Retry and Cancel buttons. |
| | |
| 16 | Stop Icon |
| 32 | Question Icon |
| 48 | Exclamation Icon |
| 64 | Information Icon |
| | |
| 0 | First button is default. |
| 256 | Second button is default. |
| 512 | Third button is default. |
| 768 | Fourth button is default |
| 0 | Application modal |
| 4096 | System modal |

The first group of values (1-5) describes the number and type of buttons displayed in the dialog box; the second group (16, 32, 48, 64) describes the icon style; the third group (0, 256, 512) determines which button is the default; and the fourth group (0, 4096) determines the modality

of the message box. When adding numbers to create a final value for the argument type, use only one number from each group. If omitted, the default value for type is 0.

The value returned by the MsgBox function indicates which button has been selected, as shown below:

| Value | Meaning |
|---|---|
| 1 | OK button selected. |
| 2 | Cancel button selected. |
| 3 | Abort button selected. |
| 4 | Retry button selected. |
| 5 | Ignore button selected. |
| 6 | Yes button selected. |
| 7 | No button selected. |

If the dialog box displays a Cancel button, pressing the Esc key has the same effect as choosing Cancel.

### MsgBox Function, MsgBox Statement Example

The example uses MsgBox to display a close without saving message in a dialog box with a Yes button a No button and a Cancel button. The Cancel button is the default response. The MsgBox function returns a value based on the button chosen by the user. The MsgBox statement uses that value to display a message that indicates which button was chosen.

```
Dim Msg, Style, Title, Help, Ctxt, Response, MyString

Msg = "Do you want to continue ?"    ' Define message
'Style = vbYesNo + vbCritical + vbDefaultButton2    ' Define buttons
Style = 4 + 16 + 256    ' Define buttons.
Title = "MsgBox Demonstration"    ' Define title
Help = "DEMO.HLP"    ' Define Help file
Ctxt = 1000     ' Define topic
   ' context.
   ' Display message.
Response = MsgBox(Msg, Style, Title, Help, Ctxt)
If Response = vbYes Then    ' User chose Yes
   MyString = "Yes"    ' Perform some action
Else     ' User chose No.
   MyString = "No"    ' Perform some action
End If
```

## MsgBox statement

Refer to MsgBox function, on page 172 for more information.

## Name statement

Name *oldname* As *newname*

Changes the name of a directory or a file.

The parameters oldname and newname are strings that can optionally contain a path.

# Now function

Now

Returns a date that represents the current date and time according to the setting of the computer's system date and time.

The Now function returns a Variant data type containing a date and time that are stored internally as a double. The number is a date and time from January 1, 100 through December 31, 9999, where January 1, 1900 is 2. Numbers to the left of the decimal point represent the date and numbers to the right represent the time.

## Example

```
Sub Main
   Dim Today
   Today = Now
End Sub
```

# Oct function

Oct( *num* )

Returns the octal value of the decimal parameter.

Oct returns a string.

## Example

```
Sub Main()
   Dim Msg, Num   ' Declare variables.
   Num = InputBox("Enter a number.")   ' Get user input.
   Msg = Num & " decimal is "
   Msg = Msg & Oct(Num) & " in octal notation."
   MsgBox Msg   ' Display results.
End Sub
```

# OKButton

OKBUTTON starting x position, starting y position, width, height.

For selecting options and closing dialog boxes.

### Example

```
Sub Main()
   Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
       TEXT 10, 10, 28, 12, "Name:"
       TEXTBOX 42, 10, 108, 12, .nameStr
       TEXTBOX 42, 24, 108, 12, .descStr
       CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
       OKBUTTON 42, 54, 40, 12
   End Dialog

   Dim Dlg1 As DialogName1
   Dialog Dlg1

   MsgBox Dlg1.nameStr
   MsgBox Dlg1.descStr
   MsgBox Dlg1.checkInt
End Sub
```

## On Error statement

`On Error { GoTo line | Resume Next | GoTo 0 }`

Sets BlueZone Basic's error-handling method and if applicable specifies the line label of the error-handling routine.

The `line` parameter refers to a label. That label must be present in the code or an error is generated.

### Example

```
Sub Main
   On Error GoTo dude   ' Turn on error checking
   Dim x as object
   x.draw   ' Object not set
   jpe   ' Undefined function call
   print 1/0   ' Division by zero
   Err.Raise 6   ' Generate an "Overflow" error
   MsgBox "Back"
   MsgBox "Jack"
   Exit Sub

dude:
   On Error Resume Next   ' turn off error checking
   MsgBox "HELLO"
   Print Err.Number, Err.Description
   Resume Next
   MsgBox "Should not get here!"
   MsgBox "What?"
   On Error Goto 0   ' turn on error checking
End Sub
```

### Errors can be raised with the syntax

`Err.Raise x`

The tables below show the corresponding descriptions for the defined values of x.

Table 21: Syntax error values

| **Value of *x*** | **Description** |
|---|---|
| 5 | “Invalid procedure call” |
| 6 | “Overflow” |
| 7 | “Out of memory” |
| 9 | “Subscript out of range” |
| 10 | “Array is fixed or temporarily locked” |
| 11 | “Division by zero” |
| 13 | “Type mismatch” |
| 14 | “Out of string space” |
| 16 | “Expression too complex” |
| 17 | “Can't perform requested operation” |
| 18 | “User interrupt occurred” |
| 20 | “Resume without error” |
| 28 | “Out of stack space” |
| 35 | “Sub, Function, or Property not defined” |
| 47 | “Too many DLL application clients” |
| 48 | “Error in loading DLL” |
| 49 | “Bad DLL calling convention” |
| 51 | “Internal error” |
| 52 | “Bad file name or number” |
| 53 | “File not found” |
| 54 | “Bad file mode” |
| 55 | “File already open” |
| 57 | “Device I/O error” |
| 58 | “File already exists” |
| 59 | “Bad record length” |
| 60 | “Disk full” |
| 62 | “Input past end of file” |
| 63 | “Bad record number” |
| 67 | “Too many files” |
| 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 or With block variable not set” |
| 92 | “For loop not initialized” |

Table 21: Syntax error values (continued)

| Value of *x* | Description |
|---|---|
| 93 | “Invalid pattern string” |
| 94 | “Invalid use of Null” |

Table 22: OLE Automation messages

| Value of *x* | Description |
|---|---|
| 429 | “OLE Automation server cannot create object” |
| 430 | “Class doesn't support OLE Automation” |
| 432 | “File name or class name not found during OLE Automation operation” |
| 438 | “Object doesn't support this property or method” |
| 440 | “OLE Automation error” |
| 443 | “OLE Automation object does not have a default value” |
| 445 | “Object doesn't support this action” |
| 446 | “Object doesn't support named arguments” |
| 447 | “Object doesn't support current local setting” |
| 448 | “Named argument not found” |
| 449 | “Argument not optional” |
| 450 | “Wrong number of arguments” |
| 451 | “Object not a collection” |

Table 23: Miscellaneous messages

| Value of *x* | Description |
|---|---|
| 444 | “Method not applicable in this context” |
| 452 | “Invalid ordinal” |
| 453 | “Specified DLL function not found” |
| 457 | “Duplicate Key” |
| 460 | “Invalid Clipboard format” |
| 461 | “Specified format doesn't match format of data” |
| 480 | “Can't create AutoRedraw image” |
| 481 | “Invalid picture” |
| 482 | “Printer error” |
| 483 | “Printer driver does not supported specified property” |
| 484 | “Problem getting printer information from the system” |

Table 24: Printer messages

| Value of *x* | Description |
|---|---|
| 485 | “Invalid picture type” |
| 520 | “Can't empty Clipboard” |
| 521 | “Can't open Clipboard” |

# Open statement

`Open` *`filename`*`$ [For` *`mode`*`] [Access` *`access`*`] As [#]`*`filenumber`*

Opens a file for input and output operations.

You must open a file before any I/O operation can be performed on it. The `Open` statement has the following parameters:

**`filename`**
File name or path.

**`mode`**
Reserved word that specifies the file mode: Append, Binary, Input, Output.

**`access`**
Reserved word that specifies which operations are permitted on the open file: Read, Write.

**`filenumber`**
Integer expression with a value between 1 and 255, inclusive. When an `Open` statement is executed, `filenumber` is associated with the file as long as it is open. Other I/O statements can use the number to refer to the file.

If file doesn't exist, it is created when a file is opened for Append, Binary or Output modes.

The argument mode is a reserved word that specifies one of the following file modes.

| Mode | Description |
|---|---|
| Input | Sequential input mode. |
| Output | Sequential output mode. |

Append Sequential output mode. Append sets the file pointer to the end of the file. A Print # or Write # statement then extends (appends to) the file.

The argument `access` is a reserved word that specifies the operations that can be performed on the opened file. If the file is already opened by another process and the specified type of access is not allowed, the Open operation fails and a Permission denied error occurs. The Access clause works only if you are using a version of MS-DOS that supports networking (MS-DOS version 3.1 or later). If you use the Access clause with a version of MS-DOS that doesn't support networking, a feature unavailable error occurs. The argument access can be one of the following reserved words.

| Access type | Description |
|---|---|
| Read | Opens the file for reading only. |
| Write | Opens the file for writing only. |
| Read Write | Opens the file for both reading and writing. This mode is valid only for Random and Binary files and files opened for Append mode. |

## Example

The following example writes data to a test file and reads it back.

```
Sub Main()
   Open "TESTFILE" For Output As #1   ' Open to write file.
   userData1$ = InputBox("Enter your own text here")
   userData2$ = InputBox("Enter more of your own text here")
   Write #1, "This is a test of the Write # statement."
   Write #1,userData1$, userData2
   Close #1

   Open "TESTFILE" for Input As #2   ' Open to read file.
   Do While Not EOF(2)
       Line Input #2, FileData   ' Read a line of data.
       PRint FileData   ' Construct message.

   Loop
   Close #2   ' Close all open files.
   MsgBox "Testing Print Statement"   ' Display message.
   Kill "TESTFILE"   ' Remove file from disk.
End Sub
```

# Option Base statement

`Option Base` *number*

Declares the default lower bound for array subscripts.

The `Option Base` statement is never required. If used, it can appear only once in a module, it can occur only in the `Declarations` section, and must be used before you declare the dimensions of any arrays.

The value of number must be either 0 or 1. The default base is 0.

The To clause in the `Dim`, `Global`, and `Static` statements provides a more flexible way to control the range of an array's subscripts. However, if you don't explicitly set the lower bound with a To clause, you can use `Option Base` to change the default lower bound to 1.

## Example

This example uses the Option Base statement to override the default base array subscript value of 0.

```
Option Base 1   ' Module level statement.
Sub Main
   Dim A(), Msg, NL   ' Declare variables.
   NL = Chr(10)   ' Define newline.
   ReDim A(20)   ' Create an array.
   Msg = "The lower bound of the A array is " & LBound(A) & "."
   Msg = Msg & NL & "The upper bound is " & UBound(A) & "."
   MsgBox Msg   ' Display message.
End Sub
```

# Option Explicit statement

`Option Explicit`

Forces explicit declaration of all variables.

The `Option explicit` statement is used outside of the script in the declarations section. This statement can be contained in a declare file or outside of any script in a file or buffer. If this statement is contained in the middle of a file the rest of the compile buffer will be affected.

### Example

```
Option Explicit
Sub Main
   Print y   ' Because y is not explicitly dimmed an error will occur.
End Sub
```

## Print # statement

Print # *filenumber*, [ [{Spc(*n*) | Tab(*n*)}][ *expressionlist*] [{; | ,}] ]

**filenumber**
Number used in an Open statement to open a sequential file. It can be any number of an open file.

**Note**

The number sign (#) preceding filenumber is not optional.

**Spc(n)**
Name of the Basic function optionally used to insert n spaces into the printed output. Multiple use is permitted.

**Tab(n)**
Name of the Basic function optionally used to tab to the nth column before printing expressionlist. Multiple use is permitted.

**expressionlist**
Numeric and/or string expressions to be written to the file.
If you omit expressionlist, the Print # statement prints a blank line in the file, but you must include the comma. Because Print # writes an image of the data to the file, you must delimit the data so it is printed correctly. If you use commas as delimiters, Print # also writes the blanks between print fields to the file.

**{;|,}**
Character that determines the position of the next character printed. A semicolon means the next character is printed immediately after the last character; a comma means the next character is printed at the start of the next print zone. Print zones begin every 14 columns. If neither character is specified, the next character is printed on the next line.

The Print # statement usually writes Variant data to a file the same way it writes any other data type. However, there are some exceptions:

- If the data being written is a Variant of VarType 0 (Empty), Print # writes nothing to the file for that data item.
- If the data being written is a Variant of VarType 1 (Null), Print # writes the literal #NULL# to the file.
- If the data being written is a Variant of VarType 7 (Date), Print # writes the date to the file using the Short Date format defined in the WIN.INI file. When either the date or the time component is missing or zero, Print # writes only the part provided to the file.

### Example 1

The following example writes data to a test file.

```
Sub Main
   Dim I, FNum, FName   ' Declare variables.
   For I = 1 To 3
      FNum = FreeFile   ' Determine next file number.
      FName = "TEST" & FNum
      Open FName For Output As FNum   ' Open file.
      Print #I, "This is test #" & I   ' Write string to file.
      Print #I, "Here is another ";  "line";  I
   Next I
   Close   ' Close all files.
End Sub
```

### Example 2

The following example writes data to a test file and reads it back.

```
Sub Main()
   Dim FileData, Msg, NL   ' Declare variables.
   NL = Chr(10)   ' Define newline.
   Open "TESTFILE" For Output As #1   ' Open to write file.
   Print #2, "This is a test of the Print # statement."
   Print #2,   ' Print blank line to file.
   Print #2, "Zone 1", "Zone 2"   ' Print in two print zones.
   Print #2, "With no space between"  ;  "."   ' Print two strings together.
   Close
   Open "TESTFILE" for Input As #2   ' Open to read file.
   Do While Not EOF(2)
      Line Input #2, FileData   ' Read a line of data.
      Msg = Msg & FileData & NL   ' Construct message.
      MsgBox Msg
   Loop
   Close   ' Close all open files.
   MsgBox "Testing Print Statement"   ' Display message.
   Kill "TESTFILE"   ' Remove file from disk.
End Sub
```

## Print method

`Print [`*`expr`*`, `*`expr`*`...]`

Prints a string to an object.

### Example

```
Sub PrintExample()
   Dim Msg, Pi   ' Declare variables.
   Let Pi = 4 * _Atn(1)   ' Calculate Pi.
   Msg = "Pi is equal to " & Str(Pi)
   MsgBox Msg   ' Display results.
   Print Pi   ' Prints the results in the compiler messages window
End Sub
```

## Randomize statement

`Randomize[`*`number`*`]`

Initializes the random number generator.

The Randomize statement has one optional parameter number. This parameter can be any valid number and is used to initialize the random number generator. If you omit the parameter then

the value returned by the Timer function is used as the default parameter to seed the random number generator.

### Example

```
Sub Main
   Dim MValue

   Randomize   ' Initialize random-number generator.
   MValue = Int((6 * Rnd) + 1)
   Print MValue
End Sub
```

## ReDim statement

ReDim *varname*(*subscripts*)[As *Type*][,*varname*(*subscripts*)]

Declares dynamic arrays and reallocate storage space.

The ReDim statement is used to size or resize a dynamic array that has already been declared using the Dim statement with empty parentheses. You can use the ReDim statement to repeatedly change the number of elements in and array but not to change the number of dimensions in an array or the type of the elements in the array.

### Example

```
Sub Main
   Dim TestArray() As Integer
   Dim I
   ReDim TestArray(10)
   For I = 1 To 10
       TestArray(I) = I + 10
       Print TestArray(I)
Next I
End Sub
```

## Rem statement

Rem *remark* ‘*remark*

Used to include explanatory remarks in a program.

The parameter remark is the text of any comment you want to include in the code.

### Example

```
Rem   This is a remark

Sub Main()
   Dim Answer, Msg   ' Declare variables.
   Do
   Answer = InputBox("Enter a value from 1 to 3.")
   Answer = 2
   If Answer >= 1 And Answer <= 3 Then   ' Check range.
      Exit Do   ' Exit Do...Loop.
   Else
       Beep   ' Beep if not in range.
   End If
   Loop
   MsgBox "You entered a value in the proper range."
End Sub
```

# Right function

Right (*stringexpression*, *n* )

Returns the right most n characters of the string parameter.

**stringexpression**
The string from which the right most characters are returned.

**n**
The number of characters that return. Must be a long integer.

## Example

```
' The example uses the Right function to return the first of two
' words input by the user.

Sub Main()
   Dim LWord, Msg, RWord, SpcPos, UsrInp   ' Declare variables.
   Msg = "Enter two words separated by a space."
   UsrInp = InputBox(Msg)   ' Get user input.
   print UsrInp
   SpcPos = InStr(1, UsrInp, " ")   ' Find space.
   If SpcPos Then
       LWord = Left(UsrInp, SpcPos - 1)   ' Get left word.
       print "LWord: ";  LWord
       RWord = Right(UsrInp, Len(UsrInp) - SpcPos)   ' Get right word.
       Msg = "The first word you entered is " & LWord
       Msg = Msg & "." & " The second word is "
       Msg = "The first word you entered is <" & LWord & ">"
       Msg = Msg & RWord & "."
   Else
       Msg = "You didn't enter two words."
   End If
MsgBox Msg   ' Display message.
End Sub
```

# RmDir statement

RmDir *path*

Removes an existing directory.

The parameter path is a string that is the name of the directory to be removed.

## Example

```
' This sample shows the functions mkdir (Make Directory)
' and rmdir (Remove Directory)

Sub Main
   Dim dirName As String

   dirName = "t1"
   mkdir dirName
   mkdir "t2"
   MsgBox "Directories: t1 and t2 created. Press OK to remove them"
   rmdir "t1"
   rmdir "t2"
End Sub
```

# Rnd function

Rnd( *number* )

Returns a random number.

The parameter number must be a valid numeric expression.

## Example

```
'Rnd Function Example

'The example uses the Rnd function to simulate rolling a pair of dice by
'generating random values from 1 to 6.  Each time this program is run,

Sub Main()
   Dim Dice1, Dice2, Msg   ' Declare variables.
   Dice1 = CInt(6 * Rnd() + 1)   ' Generate first die value.
   Dice2 = CInt(6 * Rnd() + 1)   ' Generate second die value.
   Msg = "You rolled a " & Dice1
   Msg = Msg & " and a " & Dice2
   Msg = Msg & " for a total of "
   Msg = Msg & Str(Dice1 + Dice2) & "."
   MsgBox Msg   ' Display message.
End Sub
```

# Second function

Second( *number* )

Returns an integer that is the second portion of the minute in the time parameter.

The parameter number must be a valid numeric expression.

## Example

```
' Format Function Example
' This example shows various uses of the Format function to format values
' using both named and user-defined formats.  For the date separator (/),
' time separator (:), and AM/ PM literal, the actual formatted output
' displayed by your system depends on the locale settings on which the code
' is running.  When times and dates are displayed in the development
' environment, the short time and short date formats of the code locale
' are used.  When displayed by running code, the short time and short date
' formats of the system locale are used, which may differ from the code
' locale.  For this example, English/United States is assumed.
' MyTime and MyDate are displayed in the development environment using
' current system short time and short date settings.

Sub Main

MyTime = "08:04:23 PM"
MyDate = "03/03/07"
MyDate = "January 27, 2007"

MsgBox Now
MsgBox MyTime

MsgBox Second( MyTime ) & " Seconds"
MsgBox Minute( MyTime ) & " Minutes"
MsgBox Hour( MyTime ) & " Hours"

MsgBox Day( MyDate ) & " Days"
MsgBox Month( MyDate ) & " Months"
MsgBox Year( MyDate ) & " Years"

' Returns current system time in the system-defined long time format.
MsgBox Format(Time, "Short Time")
MyStr = Format(Time, "Long Time")

' Returns current system date in the system-defined long date format.
MsgBox Format(Date, "Short Date")
MsgBox Format(Date, "Long Date")

'This section not yet supported
MsgBox Format(MyTime, "h:n:s")   ' Returns "17:4:23".
MsgBox Format(MyTime, "hh:nn:ss")   ' Returns "05:04:23".
MsgBox Format(MyDate, "dddd, mmm d yyyy")   ' Returns "Wednesday, _
                                            ' Jan 27 2007".

' If format is not supplied, a string is returned.
MsgBox Format(23)   ' Returns "23".

' User-defined formats.
MsgBox Format(5459.4, "##,##0.00")   ' Returns "5,459.40".
MsgBox Format(334.9, "###0.00")   ' Returns "334.90".
MsgBox Format(5, "0.00%")   ' Returns "500.00%".
MsgBox Format("HELLO", "<")   ' Returns "hello".
MsgBox Format("This is it", ">")   ' Returns "THIS IS IT".

End Sub
```

# Seek function

Seek( *filenumber* )

The parameter `filenumber` is used in the open statement and must be a valid numeric expression.

Seek returns a number that represents the byte position where the next operation is to take place. The first byte in the file is at position 1.

### Example

```
Sub Main
   Open "TESTFILE" For Input As #1    ' Open file for reading.
   Do While Not EOF(1)    ' Loop until end of file.
      MyChar = Input(1, #1)    ' Read next character of data.
      Print Seek(1)    ' Print byte position  .
   Loop
        Close #1    ' Close file.
End Sub
```

## Seek statement

Seek *filenumber*, *position*

The parameter filenumber is used in the open statement and must be a valid numeric expression, the parameter position is the number that indicates where the next read or write is to occur. In BlueZone Basic, position is the byte position relative to the beginning of the file.

The Seek statement sets the position in a file for the next read or write.

### Example

```
Sub Main
   Open "TESTFILE" For Input As #1    ' Open file for reading.
   For i = 1 To 24 Step 3    ' Loop until end of file.

       Seek #1, i    ' Seek to byte position
       MyChar = Input(1, #1)    ' Read next character of data.
       Print MyChar    'Print character of data
   Next i
   Close #1    ' Close file.
End Sub
```

## Select Case statement

Executes one of the statement blocks in the case based on the test variable.

```
Select Case testvar
Case var1
      Statement Block
Case var2
      Statement Block
Case Else
      Statement Block
End Select
```

The syntax supported by the Select statement includes the "To" keyword, a comma delimited list and a constant or variable.

```
Select Case Number    ' Evaluate Number.

Case 1 To 5    ' Number between 1 and 5, inclusive.

...

' The following is the only Case clause that evaluates to True.
```

```
Case 6, 7, 8   ' Number between 6 and 8.

...

Case 9 To 10   ' Number is 9 or 10.

...

Case Else   ' Other values.

...

End Select
```

### Example

```
' This rather tedious test shows nested select statements and if
' uncommented, the exit for statement.

Sub Test()
   For x = 1 to 5
      print x
      Select Case x
      Case 2
          Print "Outer Case Two"
      Case 3
          Print "Outer Case Three"
'         Exit For
          Select Case x
          Case 2
              Print "Inner Case Two"
          Case 3
              Print "Inner Case Three"
'             Exit For
          Case Else   ' Must be something else.
              Print "Inner Case Else:", x
          End Select
          Print "Done with Inner Select Case"
      Case Else   ' Must be something else.
          Print "Outer Case Else:",x
      End Select
   Next x
   Print "Done with For Loop"
End Sub
```

## SendKeys function

SendKeys( *Keys*, [*wait*] )

Sends one or more keystrokes to the active window as if they had been entered at the keyboard.

The SendKeys statement has two parameters:

**keys**
A string sent to the active window.

**wait**
Optional parameter. If omitted, it is assumed to be false. If wait is true the keystrokes must be processed before control is returned to the calling procedure.

### Example

```
Sub Main()
   Dim I, X, Msg   ' Declare variables.
   X = Shell("Calc.exe", 1)   ' Shell Calculator.
   For I = 1 To 5   ' Set up counting loop.
      SendKeys I & "{+}", True   ' Send keystrokes to Calculator
   Next I     ' to add each value of I.
   AppActivate "Calculator"   ' Return focus to Calculator.
   SendKeys "%{F4}", True   ' Alt+F4 to close Calculator.
End Sub
```

## Set statement

`Set Object = {[New] objectexpression | Nothing}`

Assigns an object to an object variable.

### Example

```
Sub Main
   Dim visio As Object
   Set visio = CreateObject( "visio.application" )
   Dim draw As Object
   Set draw = visio.Documents
   draw.Open "c:\visio\drawings\Sample1.vsd"
   MsgBox "Open docs: " & draw.Count
   Dim page As Object
   Set page = visio.ActivePage

   Dim red As Object
   Set red = page.DrawRectangle (1, 9, 7.5, 4.5)
   red.FillStyle = "Red fill"

   Dim cyan As Object
   Set cyan = page.DrawOval (2.5, 8.5, 5.75, 5.25)
   cyan.FillStyle = "Cyan fill"

   Dim green As Object
   Set green = page.DrawOval (1.5, 6.25, 2.5, 5.25)
   green.FillStyle = "Green fill"

   Dim DarkBlue As Object
   set DarkBlue = page.DrawOval (6, 8.75, 7, 7.75)
   DarkBlue.FillStyle = "Blue dark fill"

   visio.Quit
End Sub
```

## Shell function

`Shell ( app [, style] )`

Runs an executable program.

The `Shell` function has two parameters:

**app**
The name of the program to be executed. The name of the program in app must include a .PIF, .COM, .BAT, or .EXE file extension or an error occurs.

**style**
The number corresponding to the style of the window. It is also optional and if omitted the program is opened minimized with focus.
Window styles:

- Normal with focus 1,5,9
- Minimized with focus (default) 2
- Maximized with focus 3
- Normal without focus 4,8
- Minimized without focus 6,7
- Return value: ID, the task ID of the started program.

### Example

```
' This example uses Shell to leave the current application and run
' the Calculator program included with Microsoft Windows; it then
' uses the SendKeys statement to send keystrokes to add some numbers.

Sub Main()
   Dim I, X, Msg   ' Declare variables.
   X = Shell("Calc.exe", 1)   ' Shell Calculator.
   For I = 1 To 5   ' Set up counting loop.
      SendKeys I & "{+}", True   ' Send keystrokes to Calculator.
   Next I ' to add each value of I.
   AppActivate "Calculator"   ' Return focus to Calculator.
   SendKeys "%{F4}", True   ' Alt+F4 to close Calculator.
End Sub
```

## Sin function

`Sin( rad )`

Returns the sine of an angle that is expressed in radians.

### Example

```
Sub Main
   pi = 4 * Atn(1)
   rad = 90 * (pi/180)
   x = Sin(rad)
   print x
End Sub
```

## Space function

`Space[$] ( number )`

Skips a specified number of spaces in a `print#` statement.

The parameter `number` can be any valid integer and determines the number of blank spaces.

### Example

```
' This sample shows the space function

Sub Main
   MsgBox "Hello" & Space(20) & "There"
End Sub
```

# Sqr function

Sqr( *num* )

Returns the square root of a number.

The parameter num must be a valid number greater than or equal to zero.

## Example

```
Sub Form_Click()
   Dim Msg, Number   ' Declare variables.
   Msg = "Enter a non-negative number."
   Number = InputBox(Msg)   ' Get user input.
   If Number < 0 Then
       Msg = "Cannot determine the square root of a negative number."
   Else
       Msg = "The square root of " & Number & " is "
       Msg = Msg & Sqr(Number) & "."
   End If
   MsgBox Msg   ' Display results.
End Sub
```

# Static statement

Static *variable*

Declares variables and allocate storage space. These variables retain their value through the program run.

## Example

```
' This example shows how to use the static keyword to retain the value of
' the variable i in sub Joe.  If Dim is used instead of Static then i
' is empty when printed on the second call as well as the first.

Sub Main
   For i = 1 to 2
      Joe 2
   Next i
End Sub

Sub Joe( j as integer )
   Static i
   print i
   i = i + 5
   print i
End Sub
```

# Stop statement

Stop

Ends execution of the program.

The Stop statement can be placed anywhere in your code.

### Example

```
Sub Main()
   Dim x,y,z

   For x = 1 to 5
      For y = 1 to 5
           For z = 1 to 5
           Print "Looping" ,z,y,x
           Next z
       Next y
     Stop
   Next x
End Sub
```

## Str function

`Str( numericexpr )`

Returns the value of a numeric expression.

`Str` returns a String.

### Example

```
Sub Main
   Dim msg
   a = -1
   msgBox "Num = " & Str(a)
   MsgBox "Abs(Num) =" & Str(Abs(a))
End Sub
```

## StrComp function

`StrComp( nstring1,string2, [compare] )`

Returns a variant that is the result of the comparison of two strings.

### Example

```
Sub Main
   Dim MStr1, MStr2, MComp

   MStr1 = "ABCD": MStr2 = "today"    ' Define variables.
   print MStr1, MStr2
   MComp = StrComp(MStr1, MStr2)   ' Returns -1.
   print MComp
   MComp = StrComp(MStr1, MStr2)   ' Returns -1.
   print MComp
   MComp = StrComp(MStr2, MStr1)   ' Returns 1.
   print MComp
End Sub
```

## String function

`String( numeric, charcode )`

`String` returns a string.

`String` is used to create a string that consists of one character repeated over and over.

### Example

```
Sub Main
   Dim MString

   MString = String(5, "*")    ' Returns "*****".
   MString = String(5, 42)    ' Returns "44444".
   MString = String(10, "Today")    ' Returns "TTTTTTTTTT".
   Print MString
End Sub
```

## Sub statement

```
Sub SubName [(arguments)]
  Dim [variable(s)]
  [statementblock]
  [Exit Function]
End Sub
```

Declares and defines a `Sub` procedure name, parameters and code.

When the optional argument list needs to be passed the format is as follows:

```
([ByVal] variable [As type] [,ByVal] variable [As type] ]…])
```

The optional `ByVal` parameter specifies that the variable is [passed by value instead of by reference.

Refer to `ByRef` and `ByVal` in Subroutines and functions, on page 104 for more information.

The optional `As Type` parameter is used to specify the data type. Valid types are String, Integer, Double, Long, and Variant. Refer to Variable types, on page 100 for more information.

### Example

```
Sub Main
   Dim DST As String
   DST = "t1"
   mkdir DST
   mkdir "t2"
End Sub
```

## Tan function

```
Tan( angle )
```

Returns the tangent of an angle as a double.

The parameter `angle` must be a valid angle expressed in radians.

### Example

```
' This sample program show the use of the Tan function

Sub Main()
   Dim Msg, Pi    ' Declare variables.
   Pi = 4 * Atn(1)    ' Calculate Pi.
   Msg = "Pi is equal to " & Pi
   MsgBox Msg    ' Display results.
   x = Tan(Pi/4)
   MsgBox  x & " is the tangent of Pi/4"
End Sub
```

# Text statement

```
Text Starting X position, Starting Y position, width, height, label
```

Creates a text field for titles and labels.

## Example

```
Sub Main()
   Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
      TEXT 10, 10, 28, 12, "Name:"
      TEXTBOX 42, 10, 108, 12, .nameStr
      TEXTBOX 42, 24, 108, 12, .descStr
      CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
      OKBUTTON 42, 54, 40, 12
   End Dialog
   Dim Dlg1 As DialogName1
   Dialog Dlg1
   MsgBox Dlg1.nameStr
   MsgBox Dlg1.descStr
   MsgBox Dlg1.checkInt
End Sub
```

# TextBox statement

```
TextBox Starting X position, Starting Y position, width, height, default string
```

Creates a Text Box for typing in numbers and text.

## Example

```
Sub Main()
   Begin Dialog DialogName1 60, 60, 160, 70, "ASC - Hello"
      TEXT 10, 10, 28, 12, "Name:"
      TEXTBOX 42, 10, 108, 12, .nameStr
      TEXTBOX 42, 24, 108, 12, .descStr
      CHECKBOX 42, 38, 48, 12, "&CHECKME", .checkInt
      OKBUTTON 42, 54, 40, 12
   End Dialog
   Dim Dlg1 As DialogName1
   Dialog Dlg1
   MsgBox Dlg1.nameStr
   MsgBox Dlg1.descStr
   MsgBox Dlg1.checkInt
End Sub
```

# Time function

```
Time[()]
```

Returns the current system time.

## Example

```
Sub Main
   x = Time(Now)
   Print x
End Sub
```

# Timer event

`Timer`

The `Timer` event is used to track elapsed time or can be display as a stopwatch in a dialog. The timers value is the number of seconds from midnight.

Related topics: DateSerial, DateValue, Hour Minute, Now, Second TimeValue.

## Example

```
Sub Main
   Dim TS As Single
   Dim TE As Single
   Dim TEL As Single

   TS = Timer

   MsgBox "Starting Timer"

   TE = Timer

   TT = TE - TS
   Print TT
End Sub
```

# TimeSerial function

`TimeSerial( hour, minute, second )`

Returns the time serial for the supplied parameters `hour`, `minute`, and `second`.

## Example

```
Sub Main
   Dim MTime
   MTime = TimeSerial(12, 25, 27)
   Print MTime
End Sub
```

# TimeValue function

`TimeValue( TimeString )`

Returns a double precision serial number based of the supplied string parameter.

Midnight = TimeValue(“23:59:59”)

## Example

```
Sub Main
   Dim MTime
   MTime = TimeValue("12:25:27 PM")
   Print MTime
End Sub
```

# Trim functions

This topic covers the `Trim`, `LTrim` and `RTrim` functions.

```
[L | R] Trim( string )
```

LTrim, RTrim and Trim all Return a copy of a string with leading, trailing or both leading and trailing spaces removed.

LTrim, RTrim and Trim all return a string.

LTrim removes leading spaces.

RTrim removes trailing spaces.

Trim removes leading and trailing spaces.

### Example

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable.  It uses the
' Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
   MyString = "  <-Trim->  "   ' Initialize string.
   TrimString = LTrim(MyString)   ' TrimString = "<-Trim->  ".
   MsgBox "|" & TrimString & "|"
   TrimString = LCase(RTrim(MyString))   ' TrimString = "  <-trim->".
   MsgBox "|" & TrimString & "|"
   TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->".
   MsgBox "|" & TrimString & "|"
   ' Using the Trim function alone achieves the same result.
   TrimString = UCase(Trim(MyString))   ' TrimString = "<-TRIM->".
   MsgBox "|" & TrimString & "|"
End Sub
```

## Type statement

```
Type usertype   elementname As typename
      [ elementname As typename]
...
End Type
```

Defines a user-defined data type containing one or more elements.

The Type statement has these parts:

| Part | Description |
|---|---|
| Type | Marks the beginning of a user-defined type. |
| usertype | Name of a user-defined data type. It follows standard variable naming conventions. |
| elementname | Name of an element of the user-defined data type. It follows standard variable-naming conventions. |
| subscripts | Dimensions of an array element. You can declare multiple dimensions. |
| typename | One of these data types: Integer, Long, Single, Double, String (for variable-length strings), String * length (for fixed-length strings), Variant, or another user-defined type. The argument typename can't be an object type. End Type Marks the end of a user-defined type. |

Once you have declared a user-defined type using the Type statement, you can declare a variable of that type anywhere in your script. Use Dim or Static to declare a variable of a user-defined type. Line numbers and line labels aren't allowed in Type...End Type blocks.

User-defined types are often used with data records because data records frequently consist of a number of related elements of different data types. Arrays cannot be an element of a user defined type in BlueZone Basic.

### Example

```
' This sample shows some of the features of user defined types

Type type1
   a As Integer
   d As Double
   s As String
End Type

Type type2
   a As String
   o As type1
End Type

Type type3
   b As Integer
   c As type2
End Type

Dim type2a As type2
Dim type2b As type2
Dim type1a As type1
Dim type3a as type3

Sub Form_Click ()
   a = 5
   type1a.a = 7472
   type1a.d = 23.1415
   type1a.s = "YES"
   type2a.a = "43 - forty three"
   type2a.o.s = "Yaba Daba Doo"
   type3a.c.o.s = "COS"
   type2b.a = "943 - nine hundred and forty three"
   type2b.o.s = "Yogi"
   MsgBox type1a.a
   MsgBox type1a.d
   MsgBox type1a.s
   MsgBox type2a.a
   MsgBox type2a.o.s
   MsgBox type2b.a
   MsgBox type2b.o.s
   MsgBox type3a.c.o.s
   MsgBox a
End Sub
```

## UBound function

Ubound( *arrayname*[,*dimension*] )

Returns the value of the largest usable subscript for the specified dimension of an array.

### Example

```
' This example demonstrates some of the  features of arrays.  The lower
' bound for an array is 0 unless it is specified or option base is set it
' as is done in this example.

Option Base 1

Sub Main
   Dim a(10) As Double
   MsgBox "LBound: " & LBound(a) & " UBound: " & UBound(a)
   Dim i As Integer
   For i = 1 to 3
      a(i) = 2 + i
   Next i
   Print a(1),a(1),a(2), a(3)
End Sub
```

## UCase function

UCase( *string* )

Returns a copy of `string` in which all lowercase characters have been converted to uppercase.

### Example

```
' This example uses the LTrim and RTrim functions to strip leading and
' trailing spaces, respectively, from a string variable.  It  uses the
  ' Trim function alone to strip both types of spaces.
' LCase and UCase are also shown in this example as well as the use
' of nested function calls

Sub Main
   MyString = "  <-Trim->  "   ' Initialize string.
   TrimString = LTrim(MyString)   ' TrimString = "<-Trim->  ".
   MsgBox "|" & TrimString & "|"
   TrimString = LCase(RTrim(MyString))   ' TrimString = "  <-trim->".
   MsgBox "|" & TrimString & "|"
   TrimString = LTrim(RTrim(MyString))   ' TrimString = "<-Trim->".
   MsgBox "|" & TrimString & "|"
   ' Using the Trim function alone achieves the same result.
   TrimString = UCase(Trim(MyString))   ' TrimString = "<-TRIM->".
   MsgBox "|" & TrimString & "|"
End Sub
```

## Val function

Val( *string* )

Returns the numeric value of a string of characters.

### Example

```
Sub Main
   Dim Msg
   Dim YourVal As Double
   YourVal = Val(InputBox$("Enter a number"))
   Msg = "The number you entered is: " & YourVal
   MsgBox Msg
End Sub
```

# VarType function

`VarType( varname )`

Returns a value that indicates how the parameter `varname` is stored internally.

The parameter `varname` is a variant data type.

| VarType | Return value |
|---|---|
| Empty | 0 |
| Null | 1 |
| Integer | 2 |
| Long | 3 |
| Single | 4 |
| Double | 5 |
| Currency | 6 (not available at this time) |
| Date/Time | 7 |
| String | 8 |

## Example

```
If VarType(x) = 5 Then Print "Vartype is Double"   ' Display variable type.
```

# Weekday function

`Weekday( date, firstdayofweek )`

Returns an integer containing the whole number for the weekday it is representing.

## Example

```
Sub Main
   x = Weekday(#5/29/1959#)
   Print x
End Sub
```

# While...Wend statement

```
While condition
.
.
      [statements]
.
.
Wend
```

`While` begins the `While...Wend` flow of control structure. Condition is any numeric or expression that evaluates to true or false. If the condition is true the statements are executed. The statements can be any number of valid BlueZone Basic statements. `Wend` ends the `While...Wend` flow of control structure.

### Example

```
Sub Main
   Const Max = 5
   Dim A(5) As String
   A(1) = "Programmer"
   A(2) = "Engineer"
   A(3) = "President"
   A(4) = "Tech Support"
   A(5) = "Sales"
   Exchange = True

   While Exchange
      Exchange = False
      For I = 1 To Max
          MsgBox A(I)
      Next  I
   Wend
End Sub
```

## With statement

```
With object
      [statements]
End With
```

The `With` statement allows you to perform a series of commands or statements on a particular object without again referring to the name of that object. `With` statements can be nested by putting one `With` block within another `With` block. You must fully specify any object in an inner `With` block to any member of an object in an outer `With` block.

### Example

```
' This sample shows some of the features of user defined types and
' the with statement.

Type type1
   a As Integer
   d As Double
   s As String
End Type

Type type2
   a As String
   o As type1
End Type

Dim type1a As type1
Dim type2a As type2

Sub Main ()
   With type1a
       .a = 65
       .d = 3.14
   End With

   With type2a
      .a = "Hello, world"
       With .o
           .s = "Goodbye"
       End With
   End With
    type1a.s = "YES"
   MsgBox type1a.a
   MsgBox type1a.d
   MsgBox type1a.s
   MsgBox type2a.a
   MsgBox type2a.o.s
End Sub
```

## Write # statement

Write *#filenumber* [,*parameterlist* ]

Writes and formats data to a sequential file that must be opened in output or append mode.

A comma delimited list of the supplied parameters is written to the indicated file. If no parameters are present, the newline character is all that will be written to the file.

### Example

```
Sub Main()
   Open "TESTFILE" For Output As #1    ' Open to write file.
   userData1$ = InputBox("Enter your own text here")
   userData2$ = InputBox("Enter more of your own text here")
   Write #1, "This is a test of the Write # statement."
   Write #1,userData1$, userData2
   Close #1

   Open "TESTFILE" for Input As #2    ' Open to read file.
   Do While Not EOF(2)
       Line Input #2, FileData    ' Read a line of data.
       PRint FileData    ' Construct message.

   Loop
   Close #2    ' Close all open files.
   MsgBox "Testing Print Statement"    ' Display message.
   Kill "TESTFILE"    ' Remove file from disk.
End Sub
```

## Year function

Year( *serial#* )

Returns an integer representing a year between 1930 and 2029, inclusive. The returned integer represents the year of the `serial#` parameter.

The `serial#` parameter is a string that represents a date.

If `serial#` is a Null, this function returns a Null.

### Example

```
Sub Main
   MyDate = "11/11/07"
   x = Year(MyDate)
   print x
End Sub
```

# Chapter 6: BlueZone LIPI COM Object

The BlueZone LIPI Object is a Component Object Model (COM) software component for 32-bit Windows platforms. The BlueZone LIPI Object can be utilized by any COM container application like Visual Basic, Microsoft Excel, and Microsoft Word to enable communications between PCs running BlueZone iSeries Display emulation software and IBM iSeries systems. With the BlueZone LIPI Object, applications can initiate LIPI file transfers to an iSeries system.

The BlueZone LIPI Object is a language-independent software component. Programs written in Visual Basic, Pascal, C, C++, and so on can invoke the BlueZone LIPI Object to communicate with the host system. In addition, the BlueZone LIPI Object can be incorporated into many popular word processing, database, and spreadsheet macros and run by any ActiveX scripting engine, including BlueZone Script Host and Debugger.

The BlueZone LIPI Object has all of the features that are in the Host File Transfer window in the BlueZone iSeries Display. Refer to the *BlueZone Display and Printer User's Guide* for more information on the Host File Transfer window.

## BlueZone LIPI Object methods and properties

The BlueZone LIPI Object contains the following methods and properties.

### About method

The `About` method mimics the **Help** ® **About BlueZone** menu option and opens the About BlueZone window.

**Syntax**

```
object.About
```

### AddToTransferList() method

The `AddToTransferList()` method creates a new entry in the Transfer List based on the current LIPI Object settings, for example, the `LocalFile` and `RemoteFile` properties. This mimics clicking **Add To List** in the middle of the Host File Transfer window.

**Syntax**

```
object.AddToTransferList
```

### CenterTransferWindowOnOpen property

The `CenterTransferWindowOnOpen` property gets or sets the **Center Transfer Window on Open** check box in the **Options** tab of the File Options window.

**Syntax**

```
object.CenterTransferWindowOnOpen

object.CenterTransferWindowOnOpen = Value
```

**Parameters**

**`Value`**

Data type: Long

The available values are:

| Value | Description |
|---|---|
| 0 | Disable |
| 1 | Enable |

## ClientCertificate property

The `ClientCertificate` property gets or sets the **Client Certificate** options in the **Certificate** tab of the Security window.

**Syntax**

```
object.ClientCertificate

object.ClientCertificate = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | No Client Certificate |
| 1 | Client Certificate in Disk File |
| 2 | Client Certificate in Certificate Store |
| 3 | Client Certificate on Smart Card |

## ClientCertificateFileName property

The `ClientCertificateFileName` property gets or sets the **Certificate File** or **Common Name** field in the **Certificate** tab of the Security window.

**Syntax**

```
object.ClientCertificateFileName

object.ClientCertificateFileName = Value
```

**Parameters**

**`Value`**
Data type: String
The certificate or common name file name.

## CollapseListWindow method

The `CollapseListWindow` method mimics the **View ® Collapse List Window** menu option.

**Syntax**

```
object.CollapseListWindow
```

## CollapseTransferWindow method

The `CollapseTransferWindow` method mimics the **View ® Collapse Transfer Window** menu option.

**Syntax**

```
object.CollapseTransferWindow
```

## CreateiSeriesDataBaseFile method

The `CreateiSeriesDataBaseFile` method generates a new iSeries database file based on the settings passed into this method.

**Syntax**

```
object.CreateiSeriesDataBaseFile( FileName, FileType, FDFName,
DateFormat, DateSeparator, TimeFormat, TimeSeparator, DecimalSeparator,
FirstRowFieldNames, *Fields, iSeriesFile, Description)
```

**Parameters**

**`FileName`**

Data type: String

The local file name used as the source file.

**`FileType`**

Data type: Long

The type of data stored in the local PC file name. The available values are:

| Value | Description |
|---|---|
| 0 | ASCII |
| 1 | Basic Sequential |
| 2 | CSV |
| 3 | DIF |
| 4 | DOS Random |
| 5 | No Conversion |
| 6 | Tab Delimited |
| 7 | BIFF8 (XLS) |

**`FDFName`**

Data type: String

The local FDF file name that will be created.

**DateFormat**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | [DMY] Day/Month/Year |
| 1 | [EUR] European |
| 2 | [ISO] International Standards Organization |
| 3 | [JIS] JIS |
| 4 | [JUL] Julian |
| 5 | [MDY] Month/Day/Year |
| 6 | [USA] USA |
| 7 | [YMD] Year/Month/Day |
| 8 | [HMS] Hours/Minutes/Seconds |

**DateSeparator**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | [ ] Blank |
| 1 | [,] Comma |
| 2 | [.] Period |
| 3 | [/] Slash |
| 4 | [-] Dash |

**TimeFormat**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | [EUR] European |
| 1 | [HMS] Hours/Minutes/Seconds |
| 2 | [ISO] International Standards Organization |
| 3 | [JIS] JIS |
| 4 | [USA] USA |

**TimeSeparator**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | [ ] Blank |
| 1 | [,] Comma |
| 2 | [.] Period |
| 3 | [:] Colon |

**DecimalSeparator**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | [,] Comma |
| 1 | [.] Period |

**FirstRowFieldNames**
Data type: Long
Set to true if the first row of the source file includes Field Names. Not used for all file types.

**Fields**
Data type: IFields
A Fields Object that describes the field types for the source file.

**iSeriesFile**
Data type: String
The new iSeries database that will be generated on the host. For example, Library/File.

**Description**
Data type: String
Text that describes the iSeries Database on the host.

**Returns**
Data type: Long

If the method is successful, the return value is 0. To get the error message, use the `ErrorMessage` property. Refer to ErrorMessage property for more information.

## CreateiSeriesDataBaseFileWizard method

The `CreateiSeriesDataBaseFileWizard` method mimics the **Transfer ® Create iSeries Database File** menu option.

**Syntax**

```
object.CreateiSeriesDataBaseFileWizard
```

## CurrentDirectory property

The `CurrentDirectory` property gets or sets the current local working directory folder for the host file transfer.

**Syntax**

```
object.CurrentDirectory

object.CurrentDirectory = Value
```

**Parameters**

**Value**
Data type: String
The local working directory folder for the host file transfer.

# DefaultDateFormat property

The `DefaultDateFormat` property gets or sets the **Date Format** menu in the Data Options window.

**Syntax**

```
object.DefaultDateFormat

object.DefaultDateFormat = Value
```

**Parameters**

**`Value`**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Default – iSeries Job Default |
| 1 | [DMY] Day/Month/Year |
| 2 | [EUR] European |
| 3 | [ISO] International Standards Organization |
| 4 | [JIS] JIS |
| 5 | [JUL] Julian |
| 6 | [MDY] Month/Day/Year |
| 7 | [USA] USA |
| 8 | [YMD] Year/Month/Day |

# DefaultDateSeparator property

The `DefaultDateSeparator` property gets or sets the **Date Separator** menu in the Data Options window.

**Syntax**

```
object.DefaultDateSeparator

object.DefaultDateSeparator = Value
```

**Parameters**

**`Value`**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Default – iSeries Job Default |
| 1 | [ ] Blank |
| 2 | [,] Comma |
| 3 | [.] Period |
| 4 | [/] Slash |
| 5 | [-] Dash |

# DefaultDecimalSeparator property

The `DefaultDecimalSeparator` property gets or sets the **Decimal Separator** menu in the Data Options window.

**Syntax**

```
object.DefaultDecimalSeparator

object.DefaultDecimalSeparator = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Default – iSeries Job Default |
| 1 | [,] Comma |
| 2 | [.] Period |

# DefaultTimeFormat property

The `DefaultTimeFormat` property gets or sets the **Time Format** menu in the Data Options window.

**Syntax**

```
object.DefaultTimeFormat

object.DefaultTimeFormat = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Default – iSeries Job Default |
| 1 | [EUR] European |
| 2 | [HMS] Hours/Minutes/Seconds |
| 3 | [ISO] International Standards Organization |
| 4 | [JIS] JIS |
| 5 | [USA] USA |

# DefaultTimeSeparator property

The `DefaultTimeSeparator` property gets or sets the **Time Separator** menu in the Data Options window.

**Syntax**

```
object.DefaultTimeSeparator
object.DefaultTimeSeparator = Value
```

**Parameters**

**Value**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Default – iSeries Job Default |
| 1 | [ ] Blank |
| 2 | [,] Comma |
| 3 | [.] Period |
| 4 | [:] Colon |

# ErrorMessage property

The `ErrorMessage` property returns the error message of the last host transfer or action performed.

**Syntax**

```
object.ErrorMessage
```

# Fields method

The `Fields` method returns the Fields Object that is part of the BlueZone LIPI Object.

The Fields Object is used in the `CreateiSeriesDataBase` method. Refer to CreateiSeriesDataBaseFile method for more information.

**Syntax**

```
object.Fields()
```

**Example**

```
Set Fields = bzlipi.Fields()
Fields.Insert "F1", "", 1, 776, 0, "*DEFAULT", "*DEFAULT", 0, 0, 0
bzlipi.CreateiSeriesDataBaseFile( "local.txt", 0, "local.fdf", 6, 0, 4, 0, 1, 0, Fields, "MYLIB/TEST", "")
```

# HostAuthority property

The `HostAuthority` property gets or sets the **Authority** menu in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostAuthority
object.HostAuthority = Value
```

**Parameters**

**Value**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | All |
| 1 | None |
| 2 | Read |
| 3 | Read/Write |

## HostFDFFileName property

The `HostFDFFileName` property gets or sets the **FDF File Name** field in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostFDFFileName

object.HostFDFFileName = Value
```

**Parameters**

**`Value`**
Data type: String
The FDF file name.

## HostFileAction property

The `HostFileAction` property gets or sets the **File Creation** menu in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostFileAction

object.HostFileAction = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Create File and Member |
| 1 | Create Member |
| 2 | Replace Member |
| 3 | Append to Member |

## HostFileText property

The `HostFileText` property gets or sets the **File Text** field in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostFileText

object.HostFileText = Value
```

**Parameters**

**Value**
Data type: String
A description of the host file.

## HostFileType property

The `HostFileType` property gets or sets the **File Type** menu in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostFileType
object.HostFileType = Value
```

**Parameters**

**Value**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Data |
| 1 | Source |

## HostMemberText property

The `HostMemberText` property gets or sets the **Member Text** field in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostMemberText
object.HostMemberText = Value
```

**Parameters**

**Value**
Data type: String
A description of the member.

## HostPromptBeforeReplace property

The `HostPromptBeforeReplace` property gets or sets the **Prompt Before Replace** check box in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostPromptBeforeReplace
object.HostPromptBeforeReplace = Value
```

**Parameters**

**Value**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Do not prompt |
| 1 | Prompt |

## HostRecordLength property

The `HostRecordLength` property gets or sets the **Record Length** menu in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostRecordLength

object.HostRecordLength = Value
```

**Parameters**

**`Value`**
Data type: Long
Set the record length, in bytes. The available values are 1–2048.

## HostReferenceFile property

The `HostReferenceFile` property gets or sets the **Reference File** field in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostReferenceFile

object.HostReferenceFile = Value
```

**Parameters**

**`Value`**
Data type: String
Assign a reference file to the file that you are sending to the host.

## HostUseDescriptionFile property

The `HostUseDescriptionFile` property gets or sets the **Use Description File** check box in the **Host** tab of the Transfer Setup window.

**Syntax**

```
object.HostUseDescriptionFile

object.HostUseDescriptionFile = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Do not use |
| 1 | Use |

# EnableTracing property

The `EnableTracing` property gets or sets the **Enable Tracing to File** check box in the **Trace** tab of the File Options window.

**Syntax**

```
object.EnableTracing

object.EnableTracing = Value
```

**Parameters**

**`Value`**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Disable |
| 1 | Enable |

# ExpandListWindow method

The `ExpandListWindow` method mimics the **View ® Expand List Window** menu option.

**Syntax**

```
object.ExpandListWindow
```

# ExpandTransferWindow method

The `ExpandTransferWindow` method mimics the **View ® Expand Transfer Window** menu option.

**Syntax**

```
object.ExpandTransferWindow
```

# Help method

The `Help` method mimics the **Help ® BlueZone Help Topics** menu option and opens the *BlueZone Display and Printer User's Guide*.

**Syntax**

```
object.Help
```

# HostAddress property

The `HostAddress` property gets or sets the **Host Address** field in the **Connection** tab of the Transfer Setup window.

**Syntax**

```
object.HostAddress

object.HostAddress = Value
```

**Parameters**

**Value**
Data type: String
The IP address or DNS name of the host.

## InvalidCertificateAction property

The `InvalidCertificateAction` property gets or sets the **Invalid Certificates** option in the **Security** tab of the Security window.

**Syntax**

```
object.InvalidCertificateAction

object.InvalidCertificateAction = Value
```

**Parameters**

**Value**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Always Reject |
| 1 | Ask Before Accepting |
| 2 | Always Accept |

## Language property

The `Language` property gets or sets the **Language (CCSID)** menu option in the **Connection** tab of the Transfer Setup window.

**Syntax**

```
object.Language

object.Language = Value
```

**Parameters**

**Value**
Data type: String
The language to use as the coded character set identifier.

## LocalDeleteTrailingSpaces property

The `LocalDeleteTrailingSpaces` property gets or sets the **Delete Trailing Spaces** check box in the **Local** tab of the Transfer Setup window.

**Syntax**

```
object.LocalDeleteTrailingSpaces

object.LocalDeleteTrailingSpaces = Value
```

**Parameters**

**Value**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Do not delete |
| 1 | Delete |

## LocalFDFFileName property

The `LocalFDFFileName` property gets or sets the **FDF File Name** field in the **Local** tab of the Transfer Setup window.

**Syntax**

```
object.LocalFDFFileName

object.LocalFDFFileName = Value
```

**Parameters**

**Value**
Data type: String
The FDF file name.

## LocalFile property

The `LocalFile` property gets or sets the current local PC file for the host file transfer.

**Syntax**

```
object.LocalFile

object.LocalFile = Value
```

**Parameters**

**Value**
Data type: String
The local file name for the host file transfer.

## LocalIfFileExists property

The `LocalIfFileExists` property gets or sets the **If File Exists** menu in the **Local** tab of the Transfer Setup window.

**Syntax**

```
object.LocalIfFileExists

object.LocalIfFileExists = Value
```

**Parameters**

**Value**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Abort |
| 1 | Append |
| 2 | Overwrite |

## LocalPromptBeforeOverwrite property

The `LocalPromptBeforeOverwrite` property gets or sets the **Prompt Before Overwrite** check box in the **Local** tab of the Transfer Setup window.

**Syntax**

```
object.LocalPromptBeforeOverwrite

object.LocalPromptBeforeOverwrite = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Do not Prompt |
| 1 | Prompt |

## LocalSaveDescriptionFile property

The `LocalSaveDescriptionFile` property gets or sets the **Save Description File** check box in the **Local** tab of the Transfer Setup window.

**Syntax**

```
object.LocalSaveDescriptionFile

object.LocalSaveDescriptionFile = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Do not save |
| 1 | Save |

## MaxTransferListEntries property

The `MaxTransferListEntries` property gets or sets the **Max Transfer List Entries** menu in the **Options** tab of the File Options window.

**Syntax**

```
object.MaxTransferListEntries

object.MaxTransferListEntries = Value
```

**Parameters**

**Value**
Data type: Long
The available values are 1–255.

# OpenSettings method

The `OpenSettings` method opens the specified transfer file.

**Syntax**

```
object.OpenSettings( FileName )
```

**Parameters**

**FileName**
Data type: String
The file name of the iSeries Display transfer file (`*.adf`) to open. If `FileName` is blank, the Open Transfer Settings window opens.

# OutputFileType property

The `OutputFileType` gets or sets the **Output To** menu in the **Transfer** tab of the Transfer Setup window.

**Syntax**

```
object.OutputFileType

object.OutputFileType = Value
```

**Parameters**

**Value**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Display |
| 1 | File |
| 2 | Spreadsheet |

# Password property

The `Password` property gets or sets the **Password** field in the **Connection** tab of the Transfer Setup window. For security reasons, this property will always return blank.

**Syntax**

```
object.Password

object.Password = Value
```

**Parameters**

**Value**
Data type: String

## PreferredCipherSuite property

The `PreferredCipherSuite` property gets or sets the **Preferred CipherSuite** menu option in the **Security** tab of the Security window.

**Syntax**

```
object.PreferredCipherSuite

object.PreferredCipherSuite = Value
```

**Parameters**

**Value**
Data type: String

## PrivateKeyFileName property

The `PrivateKeyFileName` property gets or sets the **Private Key** or **Provider Name** field in the **Certificate** tab of the Security window.

**Syntax**

```
object.PrivateKeyFileName

object.PrivateKeyFileName = Value
```

**Parameters**

**Value**
Data type: String
The private key or provider name file name.

## ReceiveFile method

The `ReceiveFile` method receives the specified file from the host.

**Syntax**

```
object.ReceiveFile( LocalName, HostName )
```

**Parameters**

**LocalName**
Data type: String
The local file name used to receive the data from the iSeries host. If no path is given, the object uses the directory specified in the `CurrentDirectory` property. If no file name is given, the object uses the file name specified in the `LocalFile` property.

**HostName**
Data type: String
The iSeries host file used. For example, Library/File or Library/File(Member). If `HostName` is blank, the object uses the iSeries file specified in the `RemoteFile` property.

**Returns**

Data type: Long

If the transfer is successful, the return value is 0. To get the error message, use the `ErrorMessage` property. Refer to ErrorMessage property for more information.

## RemoteFile property

The `RemoteFile` property gets or sets the current host file for the host file transfer.

**Syntax**

```
object.RemoteFile

object.RemoteFile = Value
```

**Parameters**

**`Value`**
Data type: String
The remote host file for the host file transfer.

## RootCertificates property

The `RootCertificates` property gets or sets the **Root Certificates** section in the **Certificate** tab of the Security window.

**Syntax**

```
object.RootCertificates

object.RootCertificates = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Use OpenSSL Root Certificates |
| 1 | Use Windows Root Certificates |

## SaveSettings method

The `SaveSettings` method saves the specified transfer file.

**Syntax**

```
object.SaveSettings( FileName )
```

**Parameters**

**`FileName`**
Data type: String
The file name of the iSeries Display transfer file (`*.adf`) to save. If `FileName` is blank, the Save Transfer Settings As window opens.

## SaveSettingsOnClose property

The `SaveSettingsOnClose` property gets or sets the **Save Settings On Close** option in the **Options** tab of the File Options window.

**Syntax**

```
object.SaveSettingsOnClose

object.SaveSettingsOnClose = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Ask to Save Settings |
| 1 | Always Save Settings |
| 2 | Never Save Settings |

## SaveTransferListOnClose property

The `SaveTransferListOnClose` property gets or sets the **Save Transfer List On Close** option in the **Options** tab of the File Options window.

**Syntax**

```
object.SaveTransferListOnClose

object.SaveTransferListOnClose = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Ask to Save Transfer List |
| 1 | Always Save Transfer List |
| 2 | Never Save Transfer List |

## SendFile method

The `SendFile` method sends the specified local file to the host.

**Syntax**

```
object.SendFile( LocalName, HostName )
```

**Parameters**

**`LocalName`**
Data type: String
The local file name to send to the iSeries host. If no path is given, the object uses the directory specified in the `CurrentDirectory` property. If no file name is given, the object uses the file name specified in the `LocalFile` property.

**`HostName`**
Data type: String
The iSeries host file used. For example, Library/File or Library/File(Member). If `HostName` is blank, the object uses the iSeries file specified in the `RemoteFile` property.

**Returns**

Data type: Long

If the transfer is successful, the return value is 0. To get the error message, use the `ErrorMessage` property. Refer to ErrorMessage property for more information.

## SetWindowPos method

The `SetWindowPos` method changes the size and location of the Host File Transfer window.

**Syntax**

```
object.SetWindowPos( Left, Top, Width, Height )
```

**Parameters**

**`Left`**
Data type: Long
The new position of the left side of the window, in client coordinates.

**`Top`**
Data type: Long
The new position of the top of the window, in client coordinates.

**`Width`**
Data type: Long
The new width of the window, in pixels.

**`Height`**
Data type: Long
The new height of the window, in pixels.

## ShowGUI method

The `ShowGUI` method shows the Host File Transfer window. The method does not return until you close the Host Transfer window.

**Syntax**

```
object.ShowGUI
```

## ShowTransferStatusWindow property

The `ShowTransferStatusWindow` property gets or sets the **Show Transfer Status Window** check box in the **Transfer** tab of the Transfer Setup window.

**Syntax**

```
object.ShowTransferStatusWindow

object.ShowTransferStatusWindow = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Hide |
| 1 | Show |

## SQLQuery property

The `SQLQuery` property gets or sets the **SQL Query** text in the **SQL** tab of the Transfer Setup window.

**Syntax**

```
object.SQLQuery

object.SQLQuery = Value
```

**Parameters**

**`Value`**
Data type: String
A SQL query.

## SSLVersion property

The `SSLVersion` property gets or sets the **SSL Version** option in the **Security** tab of the Security window.

**Syntax**

```
object.SSLVersion

object.SSLVersion = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | SSL v3 |
| 1 | TLS v1 |

## StartTrace method

The `StartTrace` method start a LIPI trace.

**Syntax**

```
object.StartTrace( Filename )
```

**Parameters**

**`Filename`**
Data type: String
The LIPI trace file name. If the file already exists, it is overwritten.

# StopTrace method

The `StopTrace` method stops the LIPI trace if it was started.

**Syntax**

```
object.StopTrace
```

# SubmitCompletedTransfersToTheTransferList property

The `SubmitCompletedTransfersToTheTransferList` property gets or sets the **Submit Completed Transfers to the Transfer List** check box in the **Options** tab of the File Options window.

**Syntax**

```
object.SubmitCompletedTransfersToTheTransferList

object.SubmitCompletedTransfersToTheTransferList = Value
```

**Parameters**

**`Value`**

Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | Disable |
| 1 | Enable |

# TraceFileName property

The `TraceFileName` property gets or sets the **Trace File Name** field in the **Trace** tab of the File Options window.

**Syntax**

```
object.TraceFileName

object.TraceFileName = Value
```

**Parameters**

**`Value`**

Data type: String
The file name of the trace file. For example, `trace.trc`.

# TransferListAdd method

The `TransferListAdd` method mimics the **List** ® **Add** menu option and the Transfer Properties – New Entry window opens.

**Syntax**

```
object.TransferListAdd
```

## TransferListMoveDown method

The `TransferListMoveDown` method moves the specified transfer list entry down one position in the Transfer List window.

**Syntax**

```
object.TransferListMoveDown( Long Index )
```

**Parameters**

**`Index`**
Data type: Long
The Index in the Transfer List window.

## TransferListMoveUp method

The `TransferListMoveUp` method moves the specified transfer list entry up one position in the Transfer List window.

**Syntax**

```
object.TransferListMoveUp( Index )
```

**Parameters**

**`Index`**
Data type: Long
The Index in the Transfer List window.

## TransferListNew method

The `TransferListNew` method mimics the **List** ® **New** menu option except that there are no prompts.

**Syntax**

```
object.TransferListNew
```

## TransferListOpen method

The `TransferListOpen` method opens the specified transfer list.

**Syntax**

```
object.OpenSettings( FileName )
```

**Parameters**

**`FileName`**
Data type: String
The file name of the iSeries Display transfer list file (`*.adl`) to open. If `FileName` is blank, the Open Transfer List window opens.

## TransferListProperties method

The `TransferListProperties` method opens the Transfer Properties window for the specified transfer list entry.

**Syntax**

```
object.TransferListProperties( Index )
```

**Parameters**

**Index**
Data type: Long
The Index in the Transfer List window.

## TransferListRemove method

The `TransferListRemove` method deletes the specified transfer list entry.

**Syntax**

```
object.TransferListRemove( Index )
```

**Parameters**

**Index**
Data type: Long
The Index in the Transfer List window.

## TransferListRun method

The `TransferListRun` method runs the specified transfer list entry.

**Syntax**

```
object.TransferListRun( Index )
```

**Parameters**

**Index**
Data type: Long
The Index in the Transfer List window.

## TransferListSave method

The `TransferListSave` method saves the specified transfer list.

**Syntax**

```
object.TransferListSave( FileName )
```

**Parameters**

**FileName**
Data type: String
The file name of the iSeries Display transfer list file (`*.adl`) to save. If `FileName` is blank, the Save Transfer List As window opens.

## TransferMethod property

The `TransferMethod` property gets or sets the **Transfer Method** menu in the **Transfer** tab of the Transfer Setup window.

**Syntax**

```
object.TransferMethod

object.TransferMethod = Value
```

**Parameters**

**`Value`**
Data type: Long
The available values are:

| Value | Description |
|---|---|
| 0 | ASCII |
| 1 | Basic Sequential |
| 2 | CSV |
| 3 | DIF |
| 4 | DOS Random |
| 5 | No Conversion |
| 6 | Tab Delimited |
| 7 | BIFF8 |

## TransferSetup method

The `TransferSetup` method mimics the **Transfer** ® **Transfer Setup** menu option and opens the Transfer Setup window.

**Syntax**

```
object.TransferSetup
```

## UserName property

The `UserName` property gets or sets the **Username** field in the **Connection** tab of the Transfer Setup window.

**Syntax**

```
object.UserName
```

```
object.UserName = Value
```

**Parameters**

**`Value`**
Data type: String

## UseSSLEncryption property

The `UseSSLEncryption` property gets or sets the **Enable Secure Sockets Layer** check box in the **Security** tab of the Security window.

**Syntax**

```
object.UseSSLEncryption
```

```
object.UseSSLEncryption = Value
```

**Parameters**

**`Value`**
Data type: Long
The available SSL encryption values are:

| Value | Description |
|---|---|
| 0 | Disable |
| 1 | Enable |

## UseWindowsCredentials property

The `UseWindowsCredentials` property gets or sets the **Use Windows Credentials** check box in the **Connection** tab of the Transfer Setup window.

**Syntax**

```
object.UseWindowsCredentials

object.UseWindowsCredentials = Value
```

**Parameters**

**`Value`**

Data type: Long
The available Windows credential values are:

| Value | Description |
|---|---|
| 0 | Disable |
| 1 | Enable |

## Visible property

The `Visible` property gets or sets the Host File Transfer window visibility.

**Syntax**

```
object.Visible

object.Visible = Value
```

**Parameters**

**`Value`**

Data type: Long
The available window visibility values are:

| Value | Description |
|---|---|
| 0 | Hidden |
| 1 | Shown |

## WindowState property

The `WindowState` property gets or sets the state of the Host File Transfer window.

**Syntax**

```
object.WindowState

object.WindowState = Value
```

**Parameters**

**Value**
Data type: Long
The available window state values are:

| Value | Description |
|---|---|
| 0 | Restored |
| 1 | Minimized |
| 2 | Maximized |

# Fields Object methods and properties

The Fields Object contains the following methods and properties.

## Clear method

The `Clear` method deletes all of the `Fields` methods in the Field Object.

**Syntax**

```
object.Clear
```

## Count property

The `Count` property gets the current number of `Fields` methods in the Field Object.

**Syntax**

```
object.Count
```

## Delete method

The `Delete` method deletes the `Field` method specified by the `Index`.

**Syntax**

```
object.Delete( Index )
```

**Parameters**

**Index**
Data type: Long

## Field method

The `Field` method returns the field information of the `Field` specified by the `Index` parameter.

**Syntax**

```
object.Field( Index, Name, Description, Type, Length, Scale, CCSID, Default,
Allocate, Padding, NullCapable )
```

**Parameters**

**Index**
Data type: Long

**Name**
Data type: String
The name of the Field.

**Description**
Data type: String
The optional description of the field.

**Type**
Data type: Long
The field type.

**Length**
Data type: Long
The number of digits or characters for the field.

**Scale**
Data type: Long
The number of digits after the decimal.

**CCSID**
Data type: String
The host CCSID for the field.

**Default**
Data type: String
The default value of the field when it is empty.

**Allocate**
Data type: Long
The fixed length space allocated for the field for VARCHAR and VARGRAPHICS.

**Padding**
Data type: Long
Allows for extra space in a CHARACTER or GRAPHIC field.

**NullCapable**
Data type: Long
The field can contain null values.

## Insert method

The `Insert` method adds a new `Field` method to the end of the Fields Object.

**Syntax**

```
object.Insert( Name, Description, Type, Length, Scale, CCSID, Default,
Allocate, Padding, NullCapable );
```

**Parameters**

**Name**
Data type: String
The name of the `Field` method.

**Description**
Data type: String
The optional description of the field.

**Type**
Data type: Long
The field type.

**Length**
Data type: Long
The number of digits or characters for the field.

**Scale**
Data type: Long
The number of digits after the decimal.

**CCSID**
Data type: String
The host CCSID for the field.

**Default**
Data type: String
The default value of the field when it is empty.

**Allocate**
Data type: Long
The fixed length space allocated for the field for VARCHAR and VARGRAPHICS.

**Padding**
Data type: Long
Allows for extra space in a CHARACTER or GRAPHIC field.

**NullCapable**
Data type: Long
The field can contain null values.

**Example**

```
Set Fields = bzlipi.Fields() Fields.Insert "F1", "", 1, 776, 0, "*DEFAULT",
"*DEFAULT", 0, 0, 0
```

# LIPI Object examples

The following example starts a new blank session, sets options, and downloads a file from the iSeries host. A message box will open with a success or failure message.

```
set bzlipi = CreateObject("BlueZone.LIPI")
bzlipi.Username = "myuserid"
bzlipi.Password = "mypassword"
bzlipi.HostAddress = "192.168.1.100"
bzlipi.ShowTransferStatusWindow = False
bzlipi.LocalPromptBeforeOverwrite = False
result = bzlipi.ReceiveFile( "local.txt", "MYLIB/F4101" )
MsgBox bzlipi.ErrorMessage
```

The following example opens a configuration file and then starts the download from the host.

```
set bzlipi = CreateObject("BlueZone.LIPI")
bzlipi.OpenSettings "MyConfig.adf"
result = bzlipi.ReceiveFile( "local.txt", "MYLIB/F4101" )
MsgBox bzlipi.ErrorMessage
```

The following example uses the ShowGUI method. The message box will not open until the user closes the Host File Transfer window.

```
set bzlipi = CreateObject("BlueZone.LIPI")
bzlipi.ShowGUI
MsgBox "Host File Transfer has been closed"
```

# Chapter 7: BlueZone Plus VBA

Microsoft Visual Basic for Applications (VBA) is a powerful development technology for rapidly customizing Windows applications like BlueZone, and integrating them with existing data and systems.

VBA provides a complete Integrated Development Environment (IDE) that features the same elements familiar to developers using Microsoft Visual Basic, including a Project Window, a Properties Window, and debugging tools. VBA also includes support for Microsoft Forms, for creating custom dialog boxes, and ActiveX Controls, for rapidly building user interfaces. Integrated directly into BlueZone, VBA offers the advantages of fast, in-process performance, tight integration with the host application, and the ability to build solutions without the use of additional tools.

As an option, BlueZone is available with the Microsoft VBA development environment and runtime license as BlueZone Plus VBA. BlueZone Plus VBA is required for everyone who to develops and/or runs BlueZone Plus VBA applications. BlueZone Plus VBA offers a sophisticated set of programming tools based on the Microsoft Visual Basic development system.

**Note**

Existing BlueZone customers can upgrade their existing BlueZone license to BlueZone Plus VBA. Please contact your BlueZone Account Executive for more information about upgrading to BlueZone Plus VBA.

VBA is a shared component, which means you have one copy of VBA for all applications on your system that use it. Although each application uses the same Visual Basic files, the Visual Basic Editor configuration is stored separately for each product. This means that when you open the Visual Basic Editor from BlueZone, it will show your BlueZone project as you last used it, even if you have used the Visual Basic Editor for other applications.

### Differences between Visual Basic, VBA, and VBScript

**Visual Basic** is a stand-alone tool for creating separate software components, such as executable programs, COM components and ActiveX Controls and is useful when you must build a specialized solution from scratch.

**VBA** offers the same powerful tools as Visual Basic in the context of an existing application and is the best option for customizing software that already meets most of your needs.

**VBScript** is a lightweight version of the Visual Basic language and is designed specifically for use on web pages. While scripting can sometimes be used for simple automation, VBA is the premier technology designed specifically for application automation. Unlike VBA, VBScript does not have an IDE.

## BlueZone Plus VBA installation

BlueZone Plus VBA is typically installed by an administrator.

For information on installing BlueZone Plus VBA, refer to Chapter 2: Optional Features - Installing BlueZone Plus VBA. in the *BlueZone Administrator's Guide*.

## BlueZone Plus VBA Macro toolbar

The BlueZone Plus VBA Macro toolbar is optional and can be turned on or off through the **Macro** menu.

**Note**

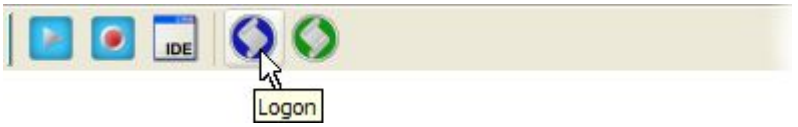
The BlueZone Plus VBA Macro toolbar can be moved around the BlueZone application window just like any Windows application by dragging the vertical bar on the left hand side of the toolbar.

By default, the BlueZone Plus VBA Macro toolbar has three items:

- Play Macro
- Record Macro
- Launch VBA IDE

### Playing a BlueZone Plus VBA Macro From the Macro toolbar

When recording macros, you have the option of creating a toolbar button. If you choose yes, buttons are automatically created on the toolbar:




**Tip**

If you place the cursor over a button, a text balloon displays the name of the macro as shown above.

To play a macro from the toolbar, click the desired macro on the toolbar.

### Customizing the BlueZone Plus VBA Macro toolbar

The BlueZone Plus VBA toolbar can be customized to suit your needs.

- Text labels located next to the tool bar items can be turned on or off.
- Macro buttons can be added or removed from the toolbar.

To toggle text labels on or off:

1. Select **Macro** ® **Properties** from the BlueZone menu bar.
2. Either select or clear the **Show Button Labels on ToolBar** check box.

To add or remove macro buttons:

1. Select **Macro** ® **Properties** from the BlueZone menu bar.
2. Click **Customize**.

Or

1. Right-click the VBA Macro toolbar.
   The Customize VBA ToolBar dialog opens.
2. Select the buttons you want to appear on the BlueZone Plus VBA Macro toolbar by selecting or clearing the corresponding items.
3. You can also **Add**, **Rename** or **Delete** macro buttons from the toolbar.

# Using BlueZone Plus VBA

In order to run BlueZone Plus VBA, you must install BlueZone Plus VBA at the same time you install BlueZone.

Refer to BlueZone Plus VBA installation, on page 232 for more information.

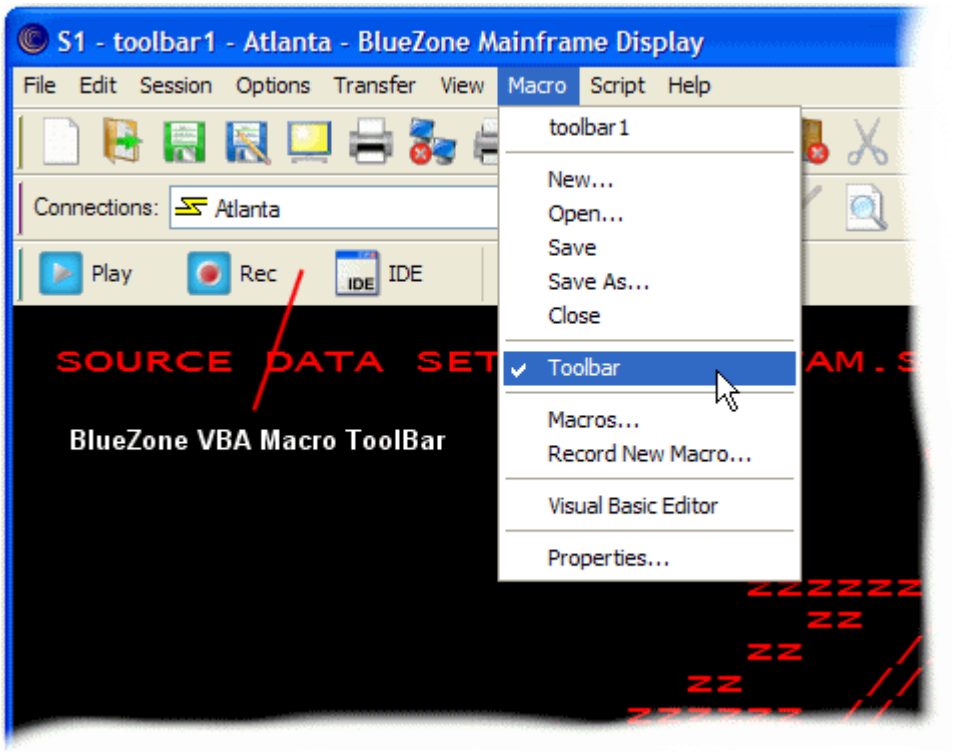
Once BlueZone and BlueZone Plus VBA have been successfully installed, BlueZone Plus VBA is accessed by selecting **Macro** from the BlueZone menu bar.

**Note**

Once BlueZone Plus VBA is installed, the native BlueZone Macro feature is disabled and replaced by BlueZone Plus VBA.

The first time you launch a BlueZone session, a VBA project (.bvp) is automatically created using the same name as the configuration file that launched the BlueZone session. A BlueZone Macro toolbar appears and the BlueZone Plus VBA menu looks similar to the one shown below:

Figure 6: BlueZone Plus VBA session display




## BlueZone Plus VBA menu items

**New**
Creates a new BlueZone Plus VBA project. You can create a project with any name you want by selecting New from the BlueZone Plus VBA Macro menu. A standard Windows file dialog opens. Type any name you want and click **Open**. A BlueZone Plus VBA project with that name is created. The new project name appears at the top of the BlueZone Plus VBA menu.

**Open**
Opens an existing BlueZone Plus VBA project. Only one BlueZone Plus VBA project can be active at a time.

**Save**
Saves the active BlueZone Plus VBA project.

**Save As**
Saves the active BlueZone Plus VBA project under a different name.

**Close**
Closes BlueZone Plus VBA. If you are not going to be using BlueZone Plus VBA for a while you can close it so it is not active.

**Toolbar**
A toggle used to either turn on or off the BlueZone Plus VBA Toolbar.

**Macros**
Displays the BlueZone Plus VBA Macro dialog which displays all of the existing macros in the currently active BlueZone Plus VBA project.

**Record New Macro**
Starts the recording of a new BlueZone Plus VBA macro.

**Visual Basic Editor**
Launches the Visual Basic Editor or sometimes called the IDE.

**Properties**
Opens the configurable properties of BlueZone Plus VBA. Currently the configurable properties of BlueZone Plus VBA are:

- **Auto-Play Macro on Connect**: If you want a macro to automatically run once the BlueZone session is connected to the host, check the check box and type the name of the macro you want to play. The path is not necessary, just the name of the macro.
- **Auto-Play Macro on Disconnect**: If you want a macro to automatically run upon disconnect, check the check box and type the name of the macro you want to play.
- **Show Button Labels on Toolbar**: This option toggles the text labels on the Toolbar on or off.

## Creating BlueZone macro projects

1. Click **Macro** ® **New** from the menu bar.
2. Type a project a name and click **Open**.
   BlueZone Plus VBA project names can contain spaces.
3. Click **Yes** to create the file.

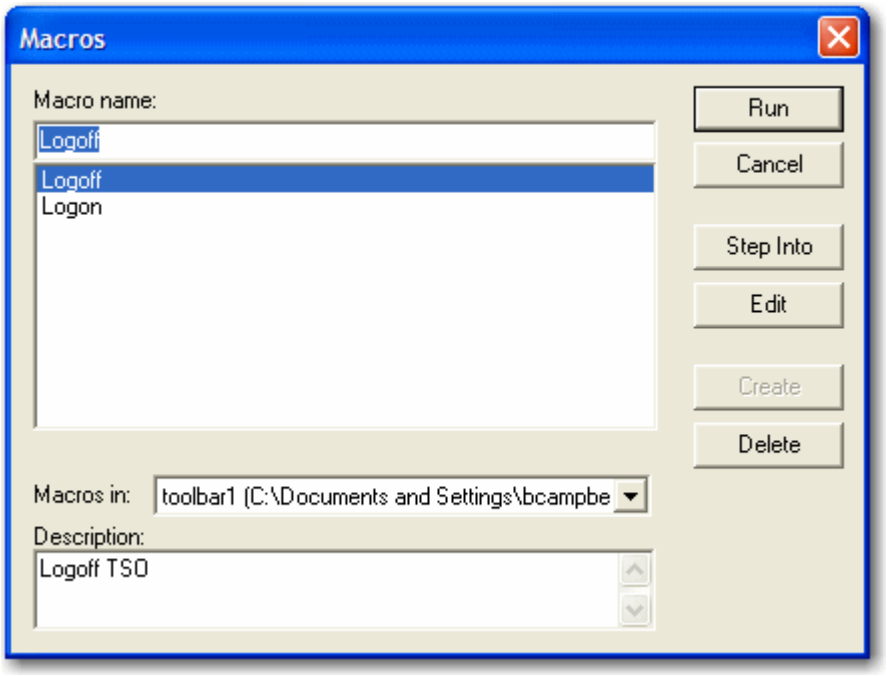
The name of the project is displayed at the top of the **Macro** menu. You can now start recording macros which are part of the newly created project.

## BlueZone Plus VBA Macro dialog

The BlueZone Plus VBA Macro dialog is used to run, debug, edit, and delete BlueZone Plus VBA macros.

To launch the BlueZone Plus VBA Macro dialog, click **Macro** ® **Macros** The Macros dialog opens:

Figure 7: Macros dialog




**Note**

Only macros that are part of the active BlueZone Plus VBA project are displayed.

## Dialog buttons

**Run**

Runs BlueZone Plus VBA macros.

To run a BlueZone Plus VBA macro select the desired macro from the list and click **Run**. The macro runs.

**Cancel**

Closes the BlueZone Plus VBA Macro dialog.

**Step Into**

Launches the Visual Basic Editor in the debug mode.

To step into a BlueZone Plus VBA macro select the desired macro from the list and click **Step Into**. The Visual Basic Editor launches in the debug mode and you can start debugging the macro.

**Edit**

Launches the Visual Basic Editor in the edit mode.

To edit a BlueZone Plus VBA macro select the desired macro from the list and click **Edit**. The Visual Basic Editor launches in the edit mode and you can start editing the macro.

**Create**

Creates a macro from scratch inside the Visual Basic Editor.

To create a macro from scratch:

1. Type the name of the macro you want to create in the **Macro name** field.
2. Click **Create**.
   The Visual Basic Editor launches.
3. You can start creating the macro.

**Delete**

Deletes a macro.

To delete a macro select the desired macro from the list and click **Delete**.

## Recording BlueZone Plus VBA macros

1. Click **Macro** ® **Record New Macro** or click the Record Macro icon  on the BlueZone Plus Macro toolbar.

   The Record Macro dialog opens.
2. In the **Macro name** field, type a name.

   Macro names cannot contain spaces.
3. Optional: In the **Description** field, type a description of the macro.
4. Check the desired **Options** check boxes.
5. Click **OK**.
6. Type the keystrokes that compose the macro.
7. When you are finished recording, click **Macro** ® **Stop Macro Recording** or click the Stop Recording macro icon  on the BlueZone Plus VBA Macro toolbar.

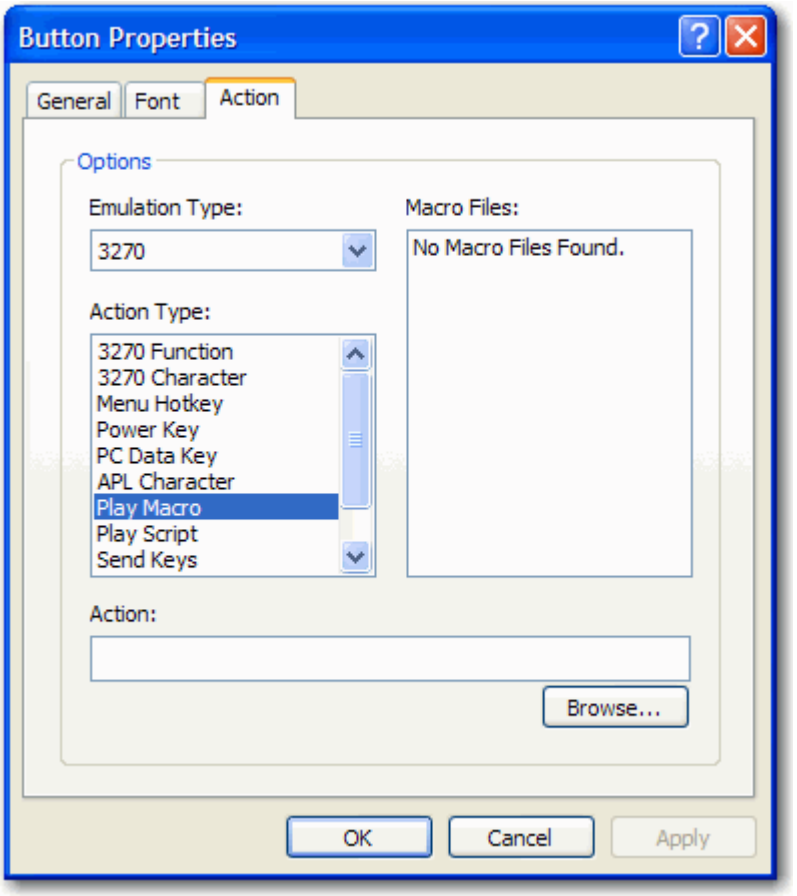
# BlueZone Plus VBA power pad buttons

In addition to being able place your VBA macros on the VBA Macro toolbar, it is also possible to create power pad buttons and assign your VBA macros to these buttons.

The process of assigning a VBA macro to a power pad button is similar to assigning a BlueZone script to a power pad button.

The difference is that when you are at the point where you assign the macro to the power pad button, the list of available VBA Macros is not displayed in the available Macro Files panel. A message appears stating that there are `No Macro files Found`:

Figure 8: Button Properties dialog




Instead, in the **Action** field, type the name of the VBA macro, with no file extension, that you want to assign to this button.

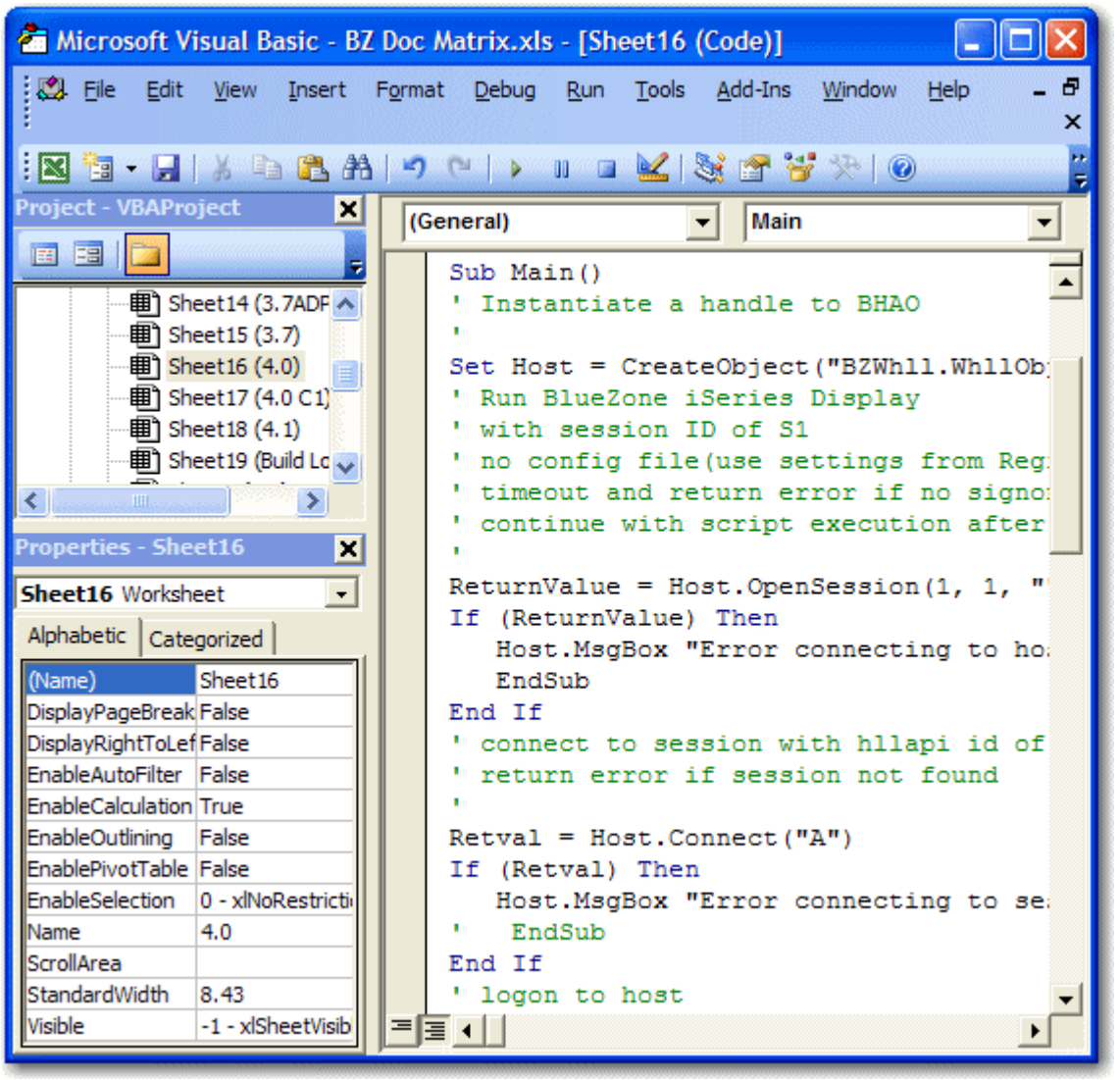
Click **OK**. The VBA macro is assigned to the power pad button.

Refer to *BlueZone Display and Printer Help* for more detailed information on the power pads feature and how to use the Power Pad Editor.

# Visual Basic Editor

The Visual Basic Editor, or sometimes referred to as the VBA IDE, can be launched from the Macro menu by selecting **Macro** ® **Visual Basic Editor** or it can be launched by clicking the VBA IDE icon  . The standard Microsoft Visual Basic Editor launches:

Figure 9: Microsoft Visual Basic Editor




Microsoft Visual Basic Help can be launched from within the Visual Basic Editor. Click **Help ® Microsoft Visual Basic Help** from the Visual Basic menu bar. Microsoft Visual basic Help launches.

# Chapter 8: Visual Basic integration

The purpose of this section is to aid the Visual Basic developer who is incorporating BlueZone into a Visual Basic application.

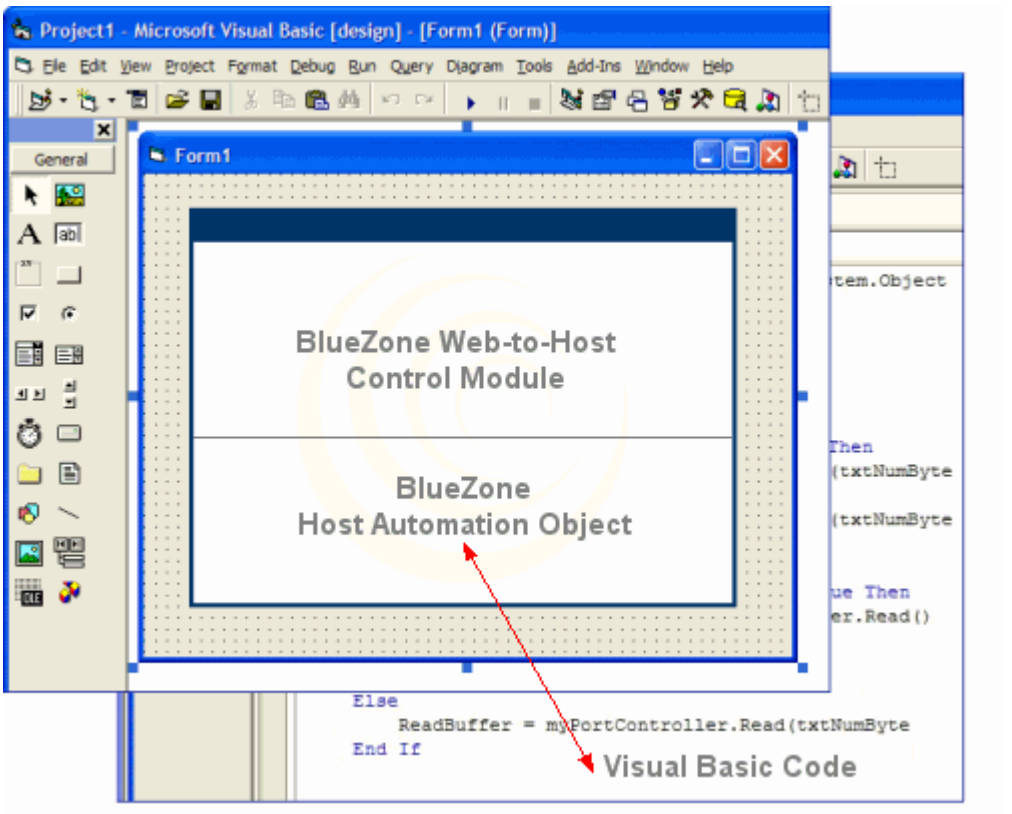
## Embed the BlueZone ActiveX control in a Visual Basic form

The BlueZone Web-to-Host control module (`sglw2hcm.ocx`) can be used to embed a BlueZone terminal emulation session in a Visual Basic form. This is useful when there is a requirement for integrating terminal emulation along with other functionality in a Visual Basic application. When the BlueZone session is running in the form, it can be controlled by the Visual Basic application through the BlueZone Host Automation Object.

Refer to BlueZone Host Automation Object Help, on page 47 for more information.

Figure 10: Visual Basic embedded with Web-to-Host control module shows the BlueZone Web-to-Host control module embedded in a Visual Basic form using the BlueZone Host Automation Object (BZHAO) to communicate with the Visual Basic code.

Figure 10: Visual Basic embedded with Web-to-Host control module




### Requirements

- Microsoft Visual Basic Editor installed
- BlueZone Desktop or BlueZone Web-to-Host installed

### Getting started

This feature is new in BlueZone 4.0. Install BlueZone using the setup program or BlueZone Web-to-Host Wizard. When installed, the BlueZone Web-to-Host control module component is available to Visual Basic.

**Note**

During the BlueZone Desktop installation, the BlueZone Web-to-Host control module file (`sglw2hcm.ocx`) is automatically copied to the BlueZone installation folder.

## Enabling the BlueZone Web-to-Host control module

1. Start a new Microsoft Visual Basic project.
2. From the menu bar, click **Project** ® **Components**.

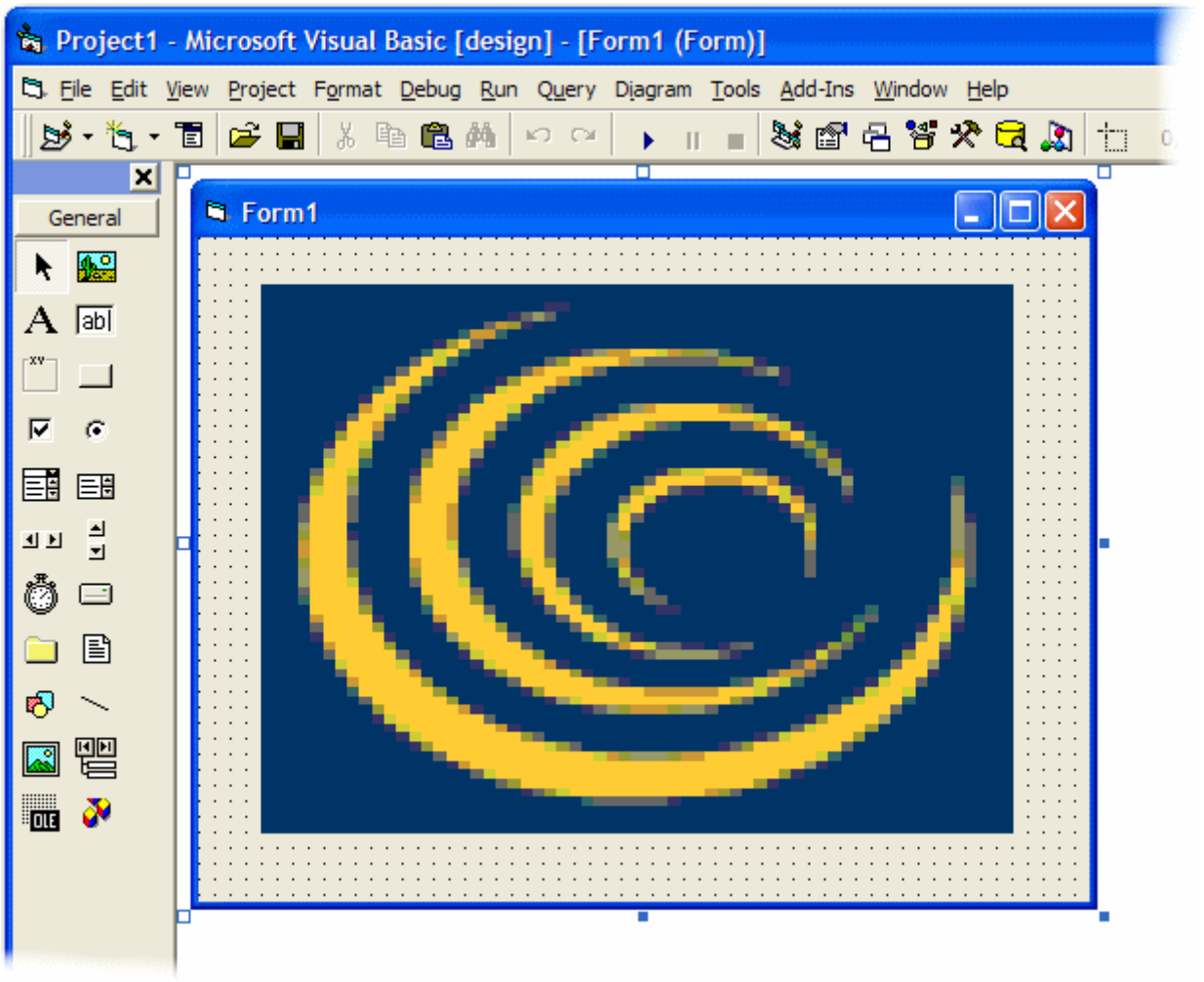
   A list of controls displays.
3. Scroll down and check the box for the **BlueZone Web-to-Host Control Module**.
4. Click **OK**.

   An icon for the BlueZone Web-to-Host control module control is added to the bottom of the Components toolbar in the Visual Basic IDE.

## Placing the BlueZone Web-to-Host control module in a form

1. If you don't already have one, add a form to your project in which to place the control.
2. Size the form to roughly the desired size. This can be changed later.
3. Select the BlueZone Web-to-Host Control component button on the Component toolbar.
4. Place the cross hair cursor on the form, click and drag the component until the desired dimensions are reached, then release. The control displays the BlueZone swirl when in design mode as shown here:

Figure 11: Web-to-Host control module in a form




## Configuring the BlueZone Web-to-Host control module

1. Right-click anywhere on the control and click **Properties**.

   A configuration dialog appears.

2. The property pages in this dialog are used to configure the control. Most of the settings can be left at the default values for the majority of applications. The **General** tab settings are as follows:

   **Local Launch**

   Instructs the Control how to launch BlueZone. You can type one of the following values:

   - `No`: Instructs BlueZone to run in Web-to-Host Mode and to use the control and files from a web server.
   - `Yes`: Instructs BlueZone to run in Web-to-Host mode and use the files in the Web to Host cache directory, without contacting the web server.
   - `Desktop`: Uses a BlueZone Desktop installation.

   **Initialization File**

   Used to specify the file containing the initialization information when running in Web-to-Host mode. Not required in Desktop mode.

**Distribution File**
Specifies the distribution file to use. Web-to-Host mode only.

**Cache Directory**
Specifies the cache directory to use. Web-to-Host mode only.

**Log Events**
Enables tracing of all of the controls functions for troubleshooting.

3. The **Sessions** tab contains settings for all of the session types supported by the BlueZone Web-to-Host control module. Only the settings for the session type specified in the **Sessions** edit box need to be completed.

   **Sessions**
   Specifies which session type is used:

   - `MD`: Mainframe Display
   - `AD`: AS400 (iSeries) Display
   - `VT`: VT Display

   **Profile**
   Complete only for the session type specified above. Type the name of the file containing the configuration settings for BlueZone.

   **Allow the User to Save Settings**
   Complete only for the session type specified above:

   - `Yes`: The user is allowed to make changes and save them to the profile.
   - `No`: The user is not allowed to make any changes to the profile.

   **Run Session in Container**
   Complete only for the session type specified above:

   - `Yes`: BlueZone is embedded in the control at the size specified by the form.
   - `No`: BlueZone runs in a separate window.
   - `Size`: BlueZone is embedded in the control at the size specified. It automatically resizes itself if the control is resized after initialization.

4. The **Launch Pad** tab enables or disables the launch pad feature. The launch pad is only used in Web-to-Host mode.
   Refer to the *BlueZone Web-to-Host Administrator's Guide* for more information.

   **Enable Launch Pad**
   Controls the launch pad feature. Type one of the following options:

   - `Yes`: Enables the launch pad, requiring user to select a BlueZone session to run.
   - `No`: Disables the launch pad, resulting in BlueZone sessions launching automatically.

   **Launch Pad Border**
   Enables or disables the border on the launch pad window.

   **Launch Pad Toolbar**
   Enables or disables the launch pad toolbar.

   **Launch Pad View**
   Sets the view for icons in the launch pad.

   **Launch Pad Buttons**
   Enables or disables the launch pad buttons.

   **Foreground Color**
   Sets the foreground color of the launch pad.

**Background Color**
Sets the background color of the launch pad.

**Watermark Image**
Specifies the file name for a bitmap used as a background image in the launch pad.

5. The **Options** tab is used to configure HTTP and Proxy settings. Unless you are using a proxy server to connect to the Internet, you probably not have to change any of these settings.

   **Cache Retries**
   Web-to-Host mode only. Leave at the default value.

   **Cache Timeout**
   Web-to-Host mode only. Leave at the default value.

   **Max HTTP 1.1 Connects**
   Sets the max number of connections the control uses to download files. Leave at the default value.

   **Max HTTP 1.0 Connects**
   Sets the max number of connections the control uses to download files. Leave at the default value.

   **Local Proxy Port**
   Sets the proxy port to access the Internet.

   **License Manager Address**
   Specifies the address for the BlueZone License Manager. This is only necessary if the address is not stored in the profile.

   **License Manager Port**
   Specifies the port for the BlueZone License Manager. This is only necessary if the port is not stored in the profile.

6. When you are finished, click **OK**. The BlueZone Web-to-Host control module is now configured.

## Communicating with BlueZone using the Host Automation Object

When BlueZone is running in the Visual Basic application, it can be controlled using the BlueZone Host Automation Object. It is instantiated using the `CreateObject` method. When created, any of the BlueZone Host Automation Object's methods and properties can be called by your Visual Basic application.

The following sample code shows the correct way to instantiate the BlueZone Host Automation Object in a VB project when the session is being embedded in a VB Form:

```
Option Explicit
Dim bzhao As Object

Private Sub Form_Unload(Cancel As Integer)
   bzhao.Disconnect
End Sub

Private Sub Sglw2hcm1_AfterConnect(ByVal Success As Long)
   Set bzhao = CreateObject("BZWhll.WhllObj")
   Bzhao.SetCursorBase 1
   bzhao.SetFormWnd Me.Hwnd
   bzhao.Connect "!"
   bzhao.WaitForText "Choice:", 23, 7, 10
   bzhao.SendKey "3"
   bzhao.SendKey "<Enter>"
bzhao.WaitReady 10, 1
End Sub
```

The above sample code is designed to work with any BlueZone session type and uses the Web-to-Host Control Module's AfterConnect event for notification that the session is ready for automation.

To use it as a test, follow these steps:

1. Start a BlueZone Display session.
2. Configure the session to auto-connect to a host system on startup.
3. In addition to host IP address and port, make any additional configuration changes at this time. All BlueZone features including the keyboard map, display colors, display font, toolbar settings, and so on are contained in the BlueZone configuration file.
4. Save the BlueZone configuration and name it, for example, `test.zmd`.
5. Create a Visual Basic Project, insert the BlueZone Web-to-Host control module in a form, and ensure that your BlueZone Web-to-Host Control Module is configured properly. Specify the configuration file name saved in the previous step as the Profile value in the Web-to-Host control's **Sessions** tab.

   Refer to the following topics for more information:

   - Enabling the BlueZone Web-to-Host control module, on page 241
   - Placing the BlueZone Web-to-Host control module in a form, on page 241
   - Configuring the BlueZone Web-to-Host control module, on page 242
6. Insert the above sample code into your Visual Basic project. Modify or remove any lines after the .Connect() method to navigate your particular host.
7. Click the Start icon  to run the application.

   BlueZone Display opens in the form, connects to the host system, and then the BZHAO is used to automate the session.
8. After the session is embedded and running in the form, you can control it using the Host Automation Object. The following sample code from above is used to create the object and connect it to the embedded session:

   ```
   Set bzhao = CreateObject("BZWhll.WhllObj")     'instantiate the BZHAO
   bzhao.SetCursorBase 1        'use 1-based row and column values
   bzhao.SetFormWnd Me.Hwnd     'set the embedded session's parent window
   bzhao.Connect "!"            'connect object to the embedded session
   ```

The above sample code demonstrates the ability for BlueZone to automatically connect to a host, wait for a specific string of characters to display on the screen, and navigate to a specific location on the host.

## CheckForIllegalCrossThreadCalls parameter

The BlueZone ActiveX control is a legacy control that was developed before .NET. It has functionality that will hook into the container window's message processing to intercept messages. Messages received are passed back to .NET, where it considers certain messages processed in this manner to be an illegal thread operation. To stop this thread warning from occurring in .NET, set the form's `CheckForIllegalCrossThreadCalls` property to `False` when initializing the form.

```
Public Class Form1
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        CheckForIllegalCrossThreadCalls = False
        InitializeComponent()

        ' Add any initialization after the InitializeComponent() call.

    End Sub
End Class
```

# BlueZone ActiveX events

**KeyDown( long *KeyCode, long Shift )**
Fires when a key is pressed down in the BlueZone session.

**KeyUp( long KeyCode, long Shift )**
Fires when a key is let up in the BlueZone session.

**AfterConnect( long Success )**
Fires after the BlueZone session connects to the host system.

**AfterDisconnect( long Success )**
Fires after the BlueZone session disconnects from the host system.

**CursorMove( long Row, long Column )**
Fires when the cursor position changes in the BlueZone session.

**ReadyToSend( BOOL State )**
Fires when the keyboard lock state changes in the BlueZone session.

**Click()**
Fires when a mouse button is clicked in the BlueZone session.

**HostScreenChange( long ScreenType, long ScreenId, long StartRow, long StartCol, long EndRow, long EndCol )**
Fires when the screen changes in the BlueZone session.

**KeyboardLock( BOOL State )**
Fires when the keyboard lock state changes in the BlueZone session.

**BeforeConnect( BOOL Cancel )**
Fires before the BlueZone session connects to the host system.

**EndOfData()**
Fires when the host system stops sending data to the BlueZone session.

**BeforeDisconnect( BOOL Cancel )**
Fires before the BlueZone session disconnects from the host system.

**Resize()**
Fires when the BlueZone session window is resized.

**KeyPress( int KeyAscii )**
Fires when a key is pressed in the BlueZone session.

**OnBlueZoneEvent( long eventNumber, String eventData )**
Fires when an event occurs that was defined using the OnEvent() method.

# Related BlueZone Host Automation object methods

**SetBrowserWnd( int BrowserWnd )**
Used to identify the browser embedded session, only valid value is window.top from web page script.

**OnEvent( int EventNum, int EventType, String Commands, int Enable, int AfterEvent, BSTR Str, int Row, int Column, int *pRetval )**
Used to define an event for use with the OnBlueZoneEvent event handler.

Refer to BlueZone Host Automation Object Help, on page 47 for more information.

# Web page script example

```
Dim bzhao

Sub Sglw2hcm1_AfterConnect( Success )
  bzhao.MsgBox "Session connected to host.", 64
End Sub

Sub Sglw2hcm1_AfterDisconnect( Success )
  bzhao.MsgBox "Session disconnected from host.", 64
End Sub

Sub Sglw2hcm1_OnBlueZoneEvent( eventNumber, eventData )
  if eventNumber = 1 then
    bzhao.MsgBox "Choice found at 23, 7", 64
  end if
End Sub

Sub Form1_OnActivate()
  set bzhao = CreateObject("BZWhll.WhllObj")
  bzhao.SetBrowserWnd window.top   ' provide browser window handle, used to identify embedded session
  bzhao.Pause 1   ' give .ocx time to launch session and attach to browser
  bzhao.setHostURL 0, 0, "locis.loc.gov"   ' set host address; 0 Mainframe Display, 0 Embedded Session, Host Address
  bzhao.connectToHost 0, 0   ' connect session to host; 0 Mainframe Display, 0 Embedded Session
  bzhao.Connect "!"   ' connect automation object to session; ! current embedded session
  bzhao.OnEvent 1, rcDisplayString, "RaiseControlEvent 1", rcEnable, rcEventReenable, "Choice", 23, 7
End Sub
```

**Note**

If using the Host Automation Object to set the host address and connect the session to the host system, then the profile (BlueZone configuration file) used to launch the session specified in the Object Tag of the web page must have the **Session** ® **Configure** ® **Auto-Connect to Host** check box disabled.

# Chapter 9: BlueZone OLE Automation

OLE (Object Linking and Embedding) is a Microsoft Windows standard for communications between applications. BlueZone can be linked to Windows applications like Microsoft Word and Microsoft Excel that support OLE.

In addition, BlueZone Host Automation Object can be used as a language-independent programmatic interface to BlueZone.

Refer to BlueZone Host Automation Object for more information.

Using OLE, it is possible to use link Microsoft Word to BlueZone in support of "mail merge" applications where the names and addresses are stored on a Mainframe (or other host) and the text of the form letters are stored and formatted in Word. You could link Word to BlueZone and "pull" the names and addresses into the Word document from the Mainframe via the OLE interface.

Also, you could link BlueZone to Microsoft Excel and have data that appears in the BlueZone emulation screen, automatically appear in an Excel spreadsheet.

## Use OLE/DDE to link BlueZone to other applications

BlueZone supports OLE/DDE links to other applications, which can be used to automate copy and paste operations.

When enabled, BlueZone establishes a "Link" with the other application. Possible uses include; automatically copying Customer information from a BlueZone emulation screen to a Microsoft Word document to perform mail merges or automatically copying data from a host screen via BlueZone, to an Excel spreadsheet. Each time the host screen changes, the data are dynamically updated in the linked application.

### Enabling linking in BlueZone

1. From the BlueZone menu bar, click **Options** ® **API**.
2. Check the **Enable DDE Server Interface** check box.
3. In the **HLLAPI Short Name Session Identifier** field, type a unique value. The default is "A". If links to multiple BlueZone sessions are required, then each session must have a unique identifier.
4. Check the **Auto-assign HLLAPI Names** check box to eliminate having to configure each session separately.
5. Select the desired data in your BlueZone emulation session to be sent to the linked application by selecting it and copying it to the Windows clipboard.

### Establishing the link in the "Linked" application

1. Launch the application that you want to "link" to BlueZone.
2. Position the cursor in the document where you want the BlueZone data to appear.
3. From the "linked" application's menu, select **Edit** ® **Paste Special**.
4. In the Paste Special window, select the **Paste link** radio button and **Unformatted Text** from the As panel.

5. Click **OK**. The "Link" is now established. Each time the BlueZone screen changes, the data is automatically updated in the "Linked" application.

# Appendix A: Reference tables

## Error codes

BlueZone error codes correspond with pre-defined Windows HLLAPI return codes.

| | | |
|---|---|---|
| WHLLOK | 0 | /* Successful. */ |
| WHLLNOTCONNECTED | 1 | /* Not Connected To Presentation Space. */ |
| WHLLBLOCKNOTAVAIL | 1 | /* Requested size is not available. */ |
| WHLLPARAMETERERROR | 2 | /* Parameter Error/Invalid Function. */ |
| WHLLBLOCKIDINVALID | 2 | /* Invalid Block ID was specified. */ |
| WHLLFTXCOMPLETE | 3 | /* File Transfer Complete. */ |
| WHLLFTXSEGMENTED | 4 | /* File Transfer Complete / segmented. */ |
| WHLLPSBUSY | 4 | /* Presentation Space is Busy. */ |
| WHLLINHIBITED | 5 | /* Inhibited/Keyboard Locked. */ |
| WHLLTRUNCATED | 6 | /* Data Truncated. */ |
| WHLLPOSITIONERROR | 7 | /* Invalid Presentation Space Position. */ |
| WHLLNOTAVAILABLE | 8 | /* Unavailable Operation. */ |
| WHLLSYSERROR | 9 | /* System Error. */ |
| WHLLWOULDBLOCK | 10 | /* Blocking error. */ |
| WHLLUNAVAILABLE | 11 | /* Resource is unavailable. */ |
| WHLLPSENDED | 12 | /* The session stopped. */ |
| WHLLUNDEFINEDKEY | 20 | /* Undefined Key Combination. */ |
| WHLLOIAUPDATE | 21 | /* OIA Updated. */ |
| WHLLPSUPDATE | 22 | /* PS Updated. */ |
| WHLLBOTHUPDATE | 23 | /* Both PS And OIA Updated. */ |
| WHLLNOFIELD | 24 | /* No Such Field Found. */ |
| WHLLNOKEYSTROKES | 25 | /* No Keystrokes are available. */ |
| WHLLPSCHANGED | 26 | /* PS or OIA changed. */ |
| WHLLFTXABORTED | 27 | /* File transfer aborted. */ |
| WHLLZEROLENFIELD | 28 | /* Field length is zero. */ |
| WHLLINVALIDTYPE | 30 | /* Invalid Cursor Type. */ |
| WHLLKEYOVERFLOW | 31 | /* Keystroke overflow. */ |
| WHLLSFACONN | 32 | /* Other application already connected. */ |
| WHLLTRANCANCLI | 34 | /* Message sent inbound to host canceled*/ |
| WHLLTRANCANCL | 35 | /* Outbound trans from host canceled. */ |
| WHLLHOSTCLOST | 36 | /* Contact with host was lost. */ |
| WHLLOKDISABLED | 37 | /* The function was successful. */ |
| WHLLNOTCOMPLETE | 38 | /* The requested fn was not completed. */ |

| | | |
|---|---|---|
| WHLLSFDDM | 39 | /* One DDM session already connected. */ |
| WHLLSFDPEND | 40 | /* Disconnected w async requests pending*/ |
| WHLLBUFFINUSE | 41 | /* Specified buffer currently in use. */ |
| WHLLNOMATCH | 42 | /* No matching request found. */ |
| WHLLLOCKERROR | 43 | /* API already locked or unlocked. */ |
| | | |
| WHLLINVALIDFUNCTIONNUM | 301 | /* Invalid function number. */ |
| WHLLFILENOTFOUND | 302 | /* File Not Found. */ |
| WHLLACCESSDENIED | 305 | /* Access Denied. */ |
| WHLLMEMORY | 308 | /* Insufficient Memory. */ |
| WHLLINVALIDENVIRONMENT | 310 | /* Invalid environment. */ |
| WHLLINVALIDFORMAT | 311 | /* Invalid format. */ |
| | | |
| WHLLINVALIDPSID | 9998 | /* Invalid Presentation Space ID. */ |
| WHLLINVALIDRC | 9999 | /* Invalid Row or Column Code. */ |
| | | |
| WHLLALREADY | 61440 | /* An async call is already outstanding */ |
| WHLLINVALID | 61441 | /* Async Task Id is invalid */ |
| WHLLCANCEL | 61442 | /* Blocking call was canceled */ |
| WHLLSYSNOTREADY | 61443 | /* Underlying subsystem not started */ |
| WHLLVERNOTSUPPORTED | 61444 | /* Application version not supported */ |

# IBM 3270/5250 send keys

Table 25: IBM 3270/5250 send keys

| Meaning | Mnemonic | Friendly Syntax | 3270 | 5250 |
|---|---|---|---|---|
| @ | @@ | | | X |
| Alt | @A | | X | |
| Alternate Cursor | @$ | | X | X |
| Attention | @A@Q | <Attn> | X | X |
| Backspace | @< | <BackSpace> | X | X |
| Backtab (Left Tab) | @B | <BackTab> | X | X |
| Clear | @C | <Clear> | X | X |
| Cmd Function Key | @A@Y | | | X |
| Cursor Down | @V | <Down> | X | X |
| Cursor Left | @L | <Left> | X | X |
| Cursor Right | @Z | <Right> | X | X |
| Cursor Select | @A@J | <CursorSelect> | X | |
| Cursor Up | @U | <Up> | X | X |

Table 25: IBM 3270/5250 send keys (continued)

| Meaning | Mnemonic | Friendly Syntax | 3270 | 5250 |
|---|---|---|---|---|
| Delete | @D | <Delete> | X | X |
| Dup | @S@x | <Dup> | X | X |
| End | @q | | | X |
| Enter | @E | <Enter> | X | X |
| Erase EOF | @F | <EraseEof> | X | X |
| Erase Input | @A@F | <EraseInput> | X | X |
| Field Exit | @A@E | <FieldExit> | | X |
| Field Mark | @S@y | <FieldMark> | X | X |
| Field - | @A@- | <FieldMinus> | | X |
| Field + | @A@+ | <FieldPlus> | | X |
| Help | @H | <Help> | | X |
| Hexadecimal | @A@X | | | X |
| Home | @0<br>(zero) | <Home> | X | X |
| Insert Mode | @I | <InsertMode> | X | X |
| Insert | @A@I | <Insert> | | X |
| Host Print | @P | | | X |
| Left | @L | <Left> | X | X |
| Left Tab (Back Tab) | @B | | X | X |
| New Line | @N | <NewLine> | X | X |
| Page Up | @u | <RollUp> | | X |
| Page Down | @v | <RollDown> | | X |
| Print (PC) | @A@t | <Print> | | X |
| Record Backspace | @A@< | | | X |
| Reset | @R | <Reset> | X | X |
| Right | @Z | <Right> | X | X |
| Right Tab (Tab) | @T | <Tab> | X | X |
| Roll Down | @v | <RollDown> | X | |
| Roll Up | @u | <RollUp> | X | |
| Shift On | @S | <ShiftOn> | X | |
| Sys Request | @A@H | <SysReq> | X | X |
| Tab (Right Tab) | @T | <Tab> | X | X |
| Test Request | @A@C | <TestRequest> | | X |
| PA1 | @x | <PA1> | X | |
| PA2 | @y | <PA2> | X | |
| PA3 | @z | <PA3> | X | |
| PA4 | @+ | | X | |

Table 25: IBM 3270/5250 send keys (continued)

| Meaning | Mnemonic | Friendly Syntax | 3270 | 5250 |
|---|---|---|---|---|
| PA5 | @% | | X | |
| PA6 | @& | | X | |
| PA7 | @' | | X | |
| PA8 | @( | | X | |
| PA9 | @) | | X | |
| PA10 | @* | | X | |
| PF1/F1 | @1 | <PF1> | X | X |
| PF2/F2 | @2 | <PF2> | X | X |
| PF3/F3 | @3 | <PF3> | X | X |
| PF4/F4 | @4 | <PF4> | X | X |
| PF5/F5 | @5 | <PF5> | X | X |
| PF6/F6 | @6 | <PF6> | X | X |
| PF7/F7 | @7 | <PF7> | X | X |
| PF8/F8 | @8 | <PF8> | X | X |
| PF9/F9 | @9 | <PF9> | X | X |
| PF10/F10 | @a | <PF10> | X | X |
| PF11/F11 | @b | <PF11> | X | X |
| PF12/F12 | @c | <PF12> | X | X |
| PF13 | @d | <PF13> | X | X |
| PF14 | @e | <PF14> | X | X |
| PF15 | @f | <PF15> | X | X |
| PF16 | @g | <PF16> | X | X |
| PF17 | @h | <PF17> | X | X |
| PF18 | @i | <PF18> | X | X |
| PF19 | @j | <PF19> | X | X |
| PF20 | @k | <PF20> | X | X |
| PF21 | @l | <PF21> | X | X |
| PF22 | @m | <PF22> | X | X |
| PF23 | @n | <PF23> | X | X |
| PF24 | @o | <PF24> | X | X |

# VT send keys

Table 26: VT send keys

| Meaning | Mnemonic | Friendly syntax |
|---|---|---|
| F1 | @1 | <F1> |
| F2 | @2 | <F2> |

Table 26: VT send keys (continued)

| Meaning | Mnemonic | Friendly syntax |
|---|---|---|
| F3 | @3 | <F3> |
| F4 | @4 | <F4> |
| F5 | @5 | <F5> |
| F6 | @6 | <F6> |
| F7 | @7 | <F7> |
| F8 | @8 | <F8> |
| F9 | @9 | <F9> |
| F10 | @a | <F10> |
| F11 | @b | <F11> |
| F12 | @c | <F12> |
| F13 | @d | <F13> |
| F14 | @e | <F14> |
| F15 | @f | <F15> |
| F16 | @g | <F16> |
| F17 | @h | <F17> |
| F18 | @i | <F18> |
| F19 | @j | <F19> |
| F20 | @k | <F20> |
| Backspace | @B | <Backspace> |
| Delete | @D | <Delete> |
| Cursor Left | @L | <CursorLeft> |
| Cursor Right | @R | <CursorRight> |
| Cursor Up | @U | <CursorUp> |
| Cursor Down | @V | <CursorDown> |
| Insert | @I | <Insert> |
| Home | @H | <Home> |
| End | @Q | <End> |
| Enter | @E | <Enter> |
| Tab | @T | <Tab> |
| Find | @F | <Find> |
| Remove | @M | <Remove> |
| Select | @C | <Select> |
| Prev Screen | @P | <PrevScreen> |
| Next Screen | @N | <NextScreen> |
| Page Up | @W | <PageUp> |
| Page Down | @X | <PageDown> |
| Answerback | @A@q | <Answerback> |

Table 26: VT send keys (continued)

| Meaning | Mnemonic | Friendly syntax |
|---|---|---|
| Escape | @A@e | <Escape> |
| Control-A(^A) | @A@A | <^a> |
| Control-B(^B) | @A@B | <^b> |
| Control-C(^C) | @A@C | <^c> |
| Control-D(^D) | @A@D | <^d> |
| Control-E(^E) | @A@E | <^e> |
| Control-F(^F) | @A@F | <^f> |
| Control-G(^G) | @A@G | <^g> |
| Control-H(^H) | @A@H | <^h> |
| Control-I(^I) | @A@I | <^i> |
| Control-J(^J) | @A@J | <^j> |
| Control-K(^K) | @A@K | <^k> |
| Control-L(^L) | @A@L | <^l> |
| Control-M(^M) | @A@M | <^m> |
| Control-N(^N) | @A@N | <^n> |
| Control-O(^O) | @A@O | <^o> |
| Control-P(^P) | @A@P | <^p> |
| Control-Q(^Q) | @A@Q | <^q> |
| Control-R(^R) | @A@R | <^r> |
| Control-S(^S) | @A@S | <^s> |
| Control-T(^T) | @A@T | <^t> |
| Control-U(^U) | @A@U | <^u> |
| Control-V(^V) | @A@V | <^v> |
| Control-W(^W) | @A@W | <^w> |
| Control-X(^X) | @A@X | <^x> |
| Control-Y(^Y) | @A@Y | <^y> |
| Control-Z(^Z) | @A@Z | <^z> |
| Control-@(^@) | @A@@ | <^@> |
| PF1 (Gold Key) | @A@1 | <PF1> |
| PF2 | @A@2 | <PF2> |
| PF3 | @A@3 | <PF3> |
| PF4 | @A@4 | <PF4> |
| Num Pad Enter | @S@E | <NumpadEnter> |
| Num Pad * | @S@* | <Numpad*> |
| Num Pad + | @S@+ | <Numpad+> |
| Num Pad / | @S@/ | <Numpad/> |
| Num Pad - | @S@- | <Numpad-> |

Table 26: VT send keys (continued)

| Meaning | Mnemonic | Friendly syntax |
|---|---|---|
| Num Pad , | @S@, | <Numpad,> |
| Num Pad . | @S@. | <Numpad.> |
| Num Pad 0 | @S@0 | <Numpad0> |
| Num Pad 1 | @S@1 | <Numpad1> |
| Num Pad 2 | @S@2 | <Numpad2> |
| Num Pad 3 | @S@3 | <Numpad3> |
| Num Pad 4 | @S@4 | <Numpad4> |
| Num Pad 5 | @S@5 | <Numpad5> |
| Num Pad 6 | @S@6 | <Numpad6> |
| Num Pad 7 | @S@7 | <Numpad7> |
| Num Pad 8 | @S@8 | <Numpad8> |
| Num Pad 9 | @S@9 | <Numpad9> |
| Alt-F1 | @Y@1 | <Alt-F1> |
| Alt-F2 | @Y@2 | <Alt-F2> |
| Alt-F3 | @Y@3 | <Alt-F3> |
| Alt-F4 | @Y@4 | <Alt-F4> |
| Alt-F5 | @Y@5 | <Alt-F5> |
| Alt-F6 | @Y@6 | <Alt-F6> |
| Alt-F7 | @Y@7 | <Alt-F7> |
| Alt-F8 | @Y@8 | <Alt-F8> |
| Alt-F9 | @Y@9 | <Alt-F9> |
| Alt-F10 | @Y@a | <Alt-F10> |
| Alt-F11 | @Y@b | <Alt-F11> |
| Alt-F12 | @Y@c | <Alt-F12> |
| Alt-F13 | @Y@d | <Alt-F13> |
| Alt-F14 | @Y@e | <Alt-F14> |
| Alt-F15 | @Y@f | <Alt-F15> |
| Alt-F16 | @Y@g | <Alt-F16> |
| Alt-F17 | @Y@h | <Alt-F17> |
| Alt-F18 | @Y@i | <Alt-F18> |
| Alt-F19 | @Y@j | <Alt-F19> |
| Alt-F20 | @Y@k | <Alt-F20> |
| Ctrl-F1 | @Z@1 | <Ctrl-F1> |
| Ctrl-F2 | @Z@2 | <Ctrl-F2> |
| Ctrl-F3 | @Z@3 | <Ctrl-F3> |
| Ctrl-F4 | @Z@4 | <Ctrl-F4> |
| Ctrl-F5 | @Z@5 | <Ctrl-F5> |

Table 26: VT send keys (continued)

| Meaning | Mnemonic | Friendly syntax |
|---|---|---|
| Ctrl-F6 | @Z@6 | <Ctrl-F6> |
| Ctrl-F7 | @Z@7 | <Ctrl-F7> |
| Ctrl-F8 | @Z@8 | <Ctrl-F8> |
| Ctrl-F9 | @Z@9 | <Ctrl-F9> |
| Ctrl-F10 | @Z@a | <Ctrl-F10> |
| Ctrl-F11 | @Z@b | <Ctrl-F11> |
| Ctrl-F12 | @Z@c | <Ctrl-F12> |
| Ctrl-F13 | @Z@d | <Ctrl-F13> |
| Ctrl-F14 | @Z@e | <Ctrl-F14> |
| Ctrl-F15 | @Z@f | <Ctrl-F15> |
| Ctrl-F16 | @Z@g | <Ctrl-F16> |
| Ctrl-F17 | @Z@h | <Ctrl-F17> |
| Ctrl-F18 | @Z@i | <Ctrl-F18> |
| Ctrl-F19 | @Z@j | <Ctrl-F19> |
| Ctrl-F20 | @Z@k | <Ctrl-F20> |

# UTS send keys

Table 27: UTS send keys

| UTS key | Mnemonic |
|---|---|
| <backspace> | @< |
| <carriagereturn> | @N |
| <clearchange> | @A@C |
| <clearpagecursorhome> | @C |
| <cleartoendoffield> | @F |
| <cleartoendofline> | @A@l |
| <cleartoendofpage> | @A@p |
| <controlpagetoggle> | @A@^ |
| <cursortoendoffield> | @A@) |
| <cursortoendofline> | @A@] |
| <cursortoendofpage> | @q |
| <cursortoeopandxmit> | @A@T |
| <cursortostartoffield> | @A@( |
| <cursortostartofline> | @A@[ |
| <deletefromline> | @D |
| <deletefrompage> | @A@% |

Table 27: UTS send keys (continued)

| UTS key | Mnemonic |
|---|---|
| <deleteline> | @A@D |
| <down> | @V |
| <duplicateline> | @A@$ |
| <F1> | @1 |
| <F2> | @2 |
| <F3> | @3 |
| <F4> | @4 |
| <F5> | @5 |
| <F6> | @6 |
| <F7> | @7 |
| <F8> | @8 |
| <F9> | @9 |
| <F10> | @a |
| <F11> | @b |
| <F12> | @c |
| <F13> | @d |
| <F14> | @e |
| <F15> | @f |
| <F16> | @g |
| <F17> | @h |
| <F18> | @i |
| <F19> | @j |
| <F20> | @k |
| <F21> | @l |
| <F22> | @m |
| <fccclear> | @A@\\ |
| <fccenable> | @A@e |
| <fccgenerate> | @A@g |
| <fcclocate> | @A@/ |
| <home> | @0 |
| <insertinpage> | @A@ |
| <insertline> | @A@L |
| <left> | @L |
| <messagewait> | @A@M |
| <pagedown> | @v |
| <pageup> | @u |
| <right> | @Z |

Table 27: UTS send keys (continued)

| UTS key | Mnemonic |
|---|---|
| <setstartofentry> | @A@S |
| <soe> | @A@S |
| <systemmode> | @A@Q |
| <tab> | @T |
| <tabback> | @B |
| <tabset> | @A@= |
| <transmit> | @E |
| <unlockkeyboard> | @R |
| <up> | @U |
| <workstationmode> | @A@W |

# Related information

You might need to refer to other sources of information when you are using BlueZone products. This section lists the documentation that supports BlueZone.

**Version 6 Release 1 product information:**

- *BlueZone Advanced Automation Developer's Guide*, BZAA-0601-DG-02
- *BlueZone Desktop Administrator's Guide*, BZD-0601-AG-03
- *BlueZone Display and Printer User's Guide*, BZDP-0601-UG-03
- *BlueZone Integration Server Administrator's Guide*, BZIS-0601-AG-01
- *BlueZone License Manager Administrator's Guide*, BZLM-0601-AG-01
- *BlueZone PasswordVault User's Guide*, BZPV-0601-UG-01
- *BlueZone Secure FTP User's Guide*, BZSF-0601-UG-01
- *BlueZone Security Sever Administrator's Guide*, BZSS-0601-AG-01
- *BlueZone Session Manager User's Guide*, BZSM-0601-UG-01
- *BlueZone Web-to-Host Administrator's Guide*, BZWH-0601-AG-01

# Index

## H



## I



## J



## L



## M

## N



## O



## P



## Q



## R



## S



## T

## U



## V



## W