Programmer's Guide and Language Reference

TTWIN BASIC

Scripting Language

email: support@ttwin.com
Web site: http://www.ttwin.com

Copyright Notice

Licenses and Trademarks

TTWIN BASIC Guide

Version 2

Table of Contents

About this Manual

This manual is designed to both introduce and provide a reference for TTWIN's scripting language, TTWIN BASIC. It is assumed that you have some knowledge of Visual Basic and techniques for programming using scripts.

Structure of this Manual

The manual is divided into three sections:

- **Programming Guide**
- **Language Reference**
- **Appendices**

These sections are organised as follows.

Programming Guide

This section provides an overview of the structure of TTWIN BASIC and informtion on specific programming tasks such as DDE.

Chapter 1	**Introduction.** A overview of TTWIN BASIC.
Chapter 2	**Variables and Expressions.** Variables, their declaration (including data types), scope and use, and expressions.

Chapter 3	**Control Structures.** A description of the control statements supported by TTWIN BASIC and their usage.
Chapter 4	**Procedures.** The use of built-in functions and subroutines, user defined functions and subroutines, the Main subroutine and external DLLs.
Chapter 5	**Dialog Boxes.** Defining and using dialog boxes.
Chapter 6	**DDE.** TTWIN BASIC's DDE client and server facilities.
Chapter 7	**Event Handling.** Creating and using event handling routines.
Chapter 8	**Run-time Error Handling.** Run-time errors and how to handle them.
Chapter 9	**Running Scripts.** Event triggered script execution.
Chapter 10	**DLL Interface.**
Chapter 11	**TTWin ActiveX Controls.**

Language Reference

This section is a complete alphabetical list of all the functions, statements and subroutines that are either unique to TTWIN BASIC or implemented from Visual Basic. This manual is not meant to replace the *Microsoft Visual Basic Language Reference* and in some cases it may be necessary to refer to the *Microsoft Visual Basic Language Reference* for more detailed information.

Appendices

This section provides additional useful information.

Appendix A	**Data Types.** The storage size and range of the supported data types.
Appendix B	**Summary.** A complete list of all the functions, keywords, subroutines and statements provided by TTWIN BASIC.
Appendix C	**Reserved Words.** TTWIN BASIC supports a large number of reserved words used by the **TTWIN** BASIC interpreter.
Appendix D	**Customer Support.** Contacting Turbosoft Support.

How to use this Manual

For the beginner, the **Programming Guide** provides an introduction to programming using TTWIN BASIC. This section is organised into chapters on specific topics, e.g., *Chapter 3 - Control Structures on page 19*, giving detailed information and examples on specific information related to a task or programming concept.

The experienced TTWIN BASIC programmer will find the **Language Reference** an invaluable reference tool. It allows the programmer to quickly access information on syntax, return values, parameters, use, examples, and related items for all functions, statements and subroutines.

If you are not sure which procedure or statement to use, first refer to the tables in *Appendix D - Summary on page 389*. This chapter contains tables of the procedures grouped by task or function. For example, all the procedures that manipulate strings are listed in the one table.

Terms

The following is a list of the terms used in this manual. TTWIN BASIC conforms closely to Visual Basic, so any terms not included in this list can be assumed to have the same definition as that found in Visual Basic.

Term	Description
comment	Comments are added to the script to explain a procedure or a particular statement. Comments, which can be added anywhere in a procedure, are ignored when the procedure executes. Comments begin with an apostrophe (**'**) or with a **Rem** statement. To append a comment to the same line as a script statement, insert an apostrophe after the statement and follow it with the comment.
expression	A combination of variables, keywords and operators. An expression is used to perform calculations and manipulate data.

Term	Description
function	A function is a *procedure* that returns a value and that can be assigned to a variable. The value is returned by assigning it to the function name in one or more statements within the function. A function is delineated by **Function** and **End Function** statements.
keyword	A word that is a recognized element of the programming language. A keyword can not be redefined by the user.
parameter	A parameter is a variable or expression passed to a procedure. The procedure can either use the variable or return a value through the parameter.
procedure	A named sequentual group of statements that is executed as a unit e.g. a *function* or *subroutine.* A procedure can take variables or expressions as *parameters* that are passed to it by a calling procedure.
reserved word	Any word, including keywords, defined by the language and therefore not available for definition by the user.

statement	A statement is a complete instruction. Statements can be:
	• **Declaration statements** - which name a variable or procedure and specify the data type.
	• **Assignments statements** - which assign a value or expression to a variable.
	• **Executable statements** -which launch an action. This includes control statement like **For** loops.
	A statement can contain keywords, operators, variables and expressions. A statement that is too long to fit on one line, can be continued onto the next line by using an underscore (_), see the following example,
	`Counter = 5`
	`While Counter`
	`Display ("FATAL ERROR. This application will abort _ in 10 seconds.")`
	`Counter = Counter - 1`
	`Wend`

Term	Description
subroutine	A subroutine is similar to a function, except that a subroutine does not return a value and nor can it be assigned to a variable. A subroutine is delineated by **Sub** and **End Sub** statements.
variable	A named storage location. Variable names must be unique within the same scope. A data type is usually specified when the variable is declared.

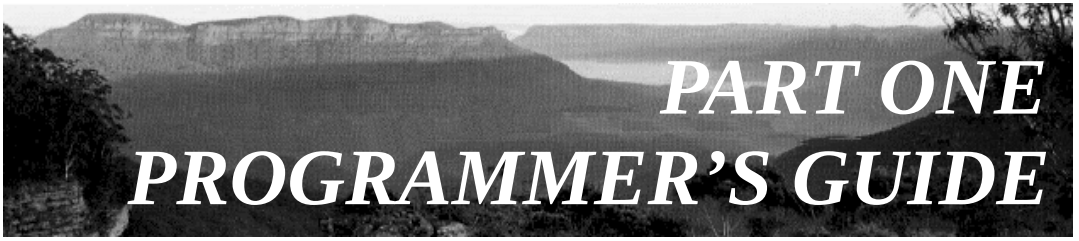
Conventions

This manual uses the following conventions:

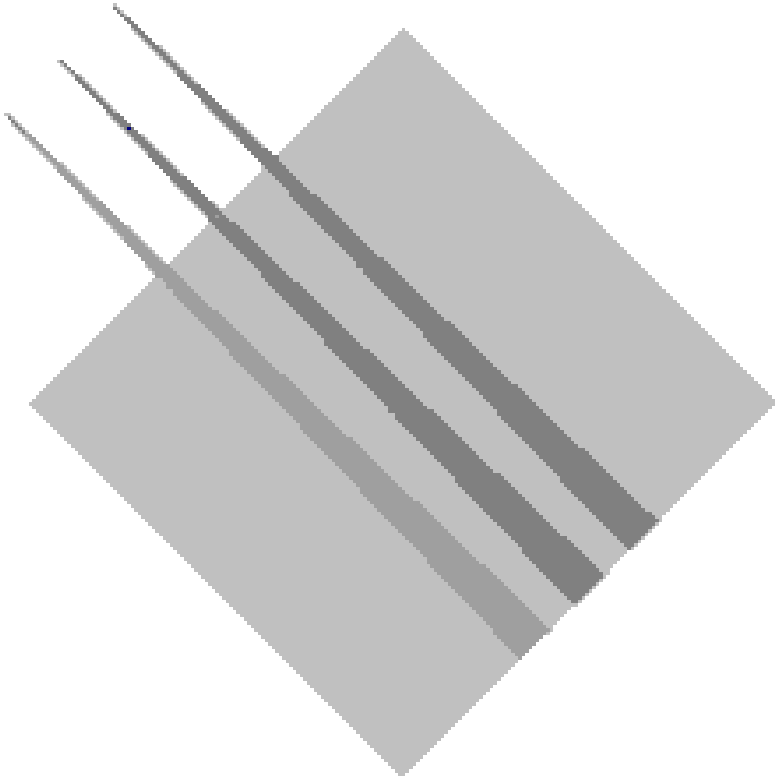
Convention	Description
If, **Dim**, **ShowMenuMode**	Keywords and reserved words used in places other than sample code, are printed in bold proportional font (Arial) using mixed case letters.
MyFile, FirstCase	In places other than examples of code, variables are printed in the italic proportional font (Times New Roman).
[*Parameters*]	In syntax, elements enclosed in square brackets are optional. For example, [*n%*] indicates *n%* is optional.
integer%	Integer variables are shown with a % symbol suffix.
string$	String variables are shown with a $ symbol suffix.
int%|*str$*	Alternative parameters are indicated by separating the possibilities with a | i.e., *n%* | *s$* indicates that the parameter can be either an integer **OR** a string.

`If 1>0 Then`
`  Display ("1>0")`
`Else`
`  Display ("Never")`
`End If`
	Examples of code are printed with as constant width font (Courier New).

Convention	Description
Actions > Disconnect	Indicates the menu option (and the menu it is on). For example, Actions > Disconnect indicates the **Disconnect** option on the **Actions** menu.
C:\SCRIPTS\TWCDLL.DLL	Words in upper case constant width font (Courier New) indicate file names and directory paths.
CTRL	All keys are shown in upper case constant width font (Courier New). (The keys on your keyboard may not be labelled exactly as they are in this manual.)
ALT_G	A sequence of simultaneously executed key strokes is indicated with an underscore. For example, ALT_F6 is the sequence generated when the ALT key is held down while F6 is pressed.



PART ONE
PROGRAMMER'S GUIDE



Chapter 1
Introduction

What is TTWIN BASIC?

TTWIN BASIC is a powerful scripting language that includes many of the functions found in other implementations of Microsoft Visual Basic as well as several functions that are unique to this implementation.

TTWIN BASIC is a comprehensible and effective tool for achieving a variety of programming objectives. Through its many functions it provides a means to:

- create and display dialog boxes
- automate login
- display buttons
- call routines in external DLLs
- print and/or clear the screen
- open and close a modem connection
- send data to and receive data from the host
- connect to and disconnect from the host
- control TTWIN's DDE client/server features

as well as many other facilities.

These facilities offer the means to enhance legacy applications with dialog boxes, scroll bars and automatic reformatting of data amongst many other possibilities.

When implementing TTWIN's scripting language, Turbosoft chose to closely follow the format of Visual Basic in order to make it possible to script without the need to learn yet another language.

Experienced programmers will have the choice of using well known functions, or finding new and innovative solutions in a language that has the traditional simplicity of generic Basic.

Creating and Editing TTWIN BASIC

To create, run, edit, compile and verify TTWIN BASIC script select the **Run/ Edit...** option on the **Scripts** menu. This option is described in detail in *Chapter 8 - Scripts Menu* of the *TTWIN User's Guide*.

Online Help

TTWIN BASIC comes with extensive online help that provides assistance at the click of a button. Scripting language help can be accessed in two ways:

- By selecting **Index** from the **Help** menu and then selecting **TTWin Programming Language**.
- By clicking on the **Help** button on the **Scripts** tile and then choosing from the **Table of contents**. (The **Scripts** tile is displayed when you select **Run/Edit...** from the **Scripts** menu.)

As well as help on the TTWIN BASIC procedures, keywords and statements, there are also sample scripts and information on using scripts. To access this material, run the script **TTWin Scripting Examples** available from the **Scripts** menu.

Chapter 2
Variables and Expressions

This chapter covers data types, declaring constants and variables, scope and using expressions.

Data Types

TTWIN's data type support is based on a subset of the Visual Basic implementation, with the exception of the exclusions listed below.

Implemented Data Types	Non implemented Data Types
Double	**Boolean**
Integer	**Byte**
Long	**Currency**
Single	**Date**
String	**Object**
Variant	

Within variant, the following subtypes are supported:

Implemented Variant Subtypes	Non implemented Variant Subtypes
Double	**Boolean**
Integer	**Byte**
Long	**Currency**
Single	**Date**
String	**Object**
Empty	
Null	

Please refer to *Appendix A - Data Type Definitions on page 377* for data type definitions and range.

Note: *Constants are **NOT** supported.*

Declaring Variables

To declare variables within a subroutine, function or main body of a program the **Dim** statement is used. The **Dim** statement is used to declare all types of variables including arrays (see *Declaring Arrays on page 16*). The syntax is,

Dim *VarName* [,*VarName...*] [**As** *Type*]

where *VarName* is a user defined name for a variable, **As** is a keyword and *Type* is one of the standard TTWIN BASIC types. When *Type* is not specified, **Integer** is assumed.

Note: *User defined types are **NOT** supported.*

Implicit Declaration

You can choose not to declare a variable before using it. In this instance TTWIN BASIC will automatically create the variable.

For example,

```
Function Cubed(Param)
     x = Abs(Param)
     Cubed = Sqr(x) * x
End Function
```

TTWIN BASIC automatically creates the variable, *x*. This is convenient, but it can lead to errors as a result of misspelling or misnaming variables, and it is not recommended.

Explicit Declaration

To overcome the problems caused by implicit variable decalrations you can use the **Option Explicit** statement to force the declaration of all variables.

The above example becomes,

```
Option Explicit
...
...
Function Cubed(Param)
     Dim x As Integer
     x = Abs(Param)
     Cubed = Sqr(x) * x
End Function
```

If the **Dim** statement in the above example was omitted or if a variable name was misspelt, an undeclared variable error would occur.

Declaring Arrays

Arrays can be one dimensional or multi-dimensional and are also declared with the **Dim** statement. The syntax is,

Dim *ArrayName* ([*LowerLimit* **To**] *UpperLimit*[,[*LowerLimit* **To**] *UpperLimit*...]) [**As** *Type*]

where *ArrayName* is a user defined name for an array variable, *LowerLimit* and *UpperLimit* are the upper and lower bounds of an array, **To** and **As** are keywords and *Type* is one of the standard TTWIN BASIC types.

When *LowerBound* is not specified, a default of 0 is used. *UpperBound* is the number of elements in the array only when *LowerBound* is 1. When *Type* is not specified, **Integer** is assumed.

For example,

```
Dim MultiArray (1 To 5, 1 to 10) As Integer
```

declares a multi-dimensional array, *MultiArray*, of integers, 5 elements by 10 elements.

Dynamic Arrays

A dynamic array can be re-dimensioned at any time. This is particularly useful when you don't know the size of the array at the time of the declaration. To create a dynamic array, first, using either the **Dim** statement or the **Static** statement (see *Static Variables on page 17*)declare the array as follows:

```
Dim DynArray () As Integer
```

Then later allocate the space for the array using the **ReDim** statment.

The syntax for the **ReDim** is the same as the **Dim** statement. For example,

```
ReDim DynArray (1, 9) As Integer
```

This statement allocates space for a multi-dimension integer array of 2 by 10 elements.

Declaring Strings

Strings are declared in the same way as any variable, except that fixed string declarations are **NOT** supported. So for example, the following declaration would **NOT** be permitted:

```
Dim MyVar As String * 10   ' fixed sting of size 10
```


The Scope of a Variable

All variables, including arrays, declared explicitly using the **Dim** statement are **local** or temporary variables. This also applies to variables that are defined implicitly (automatic variable creation).

A **local** variable only exists while the procedure, within which it was defined, is active, and hence its value is not retained between calls. If the procedure is called recursively then there will be two separate instances of each local variable.

Note: *The use of the keyword,* ***Global****, with variable declarations is* ***NOT*** *supported.*

Static Variables

To retain the value of a local variable between calls the **Static** statement is used. Variables declared as **Static** exist for the entire time that your application is executing and there is only ever one instance of them.

A function or subroutine can also be declared with **Static** in which case all the variables within the procedure are static.

```
Sub MySub
     Dim State As Integer
     Static New As Integer
     ...
```

In the above example, *State* is tempory and *New* is static.

```
Static Sub MySub
     Dim State As Integer
     Static New As Integer
     ...
```

In this example both variables are static since the subroutine has been declared as static.

Expressions

Expressions in TTWIN BASIC are formed in the standard way.

- Operator precedence is correctly handled and bracketing of expressions is fully supported.
- Strings can be concatenated with '+'.
- Integers & strings can use the following comparison operators:

  `< > <= >= <> =`
- Integer expressions can use:

  `- + / * ^`
- Boolean expressions can use:

  And Or Not
- Division of a number returning only the remainder:

  Mod

Chapter 3
Control Structures

Program flow in TTWIN BASIC, as in Visual Basic, is controlled by flow control statements. By using these statements the sequence of instructions can be dynamically changed.

The standard Visual Basic control statements implemented in TTWIN BASIC are as follows: (The syntax used in this chapter is described in full in *Conventions on page 5*.)

- **Do...Loop**
- **For...Next**
- **GoTo**
- **If...Then...Else**
- **If...Then...ElseIf**
- **Select Case**
- **While**

The **For Each...Next** statements has **NOT** been implemented but the same outcome can be achieved with a **For...Next** statement.

Do...Loop

The **Do...Loop** repeats a group of statements based on a condition. The loop can be either executed while a condition is true or until a condition becomes true. There are 2 forms of the **Do...Loop** with the syntax that follows:

Do [**While** | **Until** *Condition*]

[*Statements*]

Loop

OR

Do

[*Statements*]

Loop [**While** | **Until** *Condition*]

where *Condition* is any numeric or string expression, and *Statements* is any number of valid TTWIN BASIC statement. Within *Statements* can also be an **Exit Do** statement as an alternate way to exit the **Do...Loop**. Any number of **Exit Do** statements can be placed anywhere in *Statements.*

The **Exit Do** statement transfers control to the statement immediately following the **Loop** statement. Therefore, when used within nested **Do...Loop** statements, an **Exit Do** transfers control to the loop that is one level above.

For example, the following loop is executed 10 times,

```
Count = 0
Do
     ...
     Count = Count + 1
Loop Until Count = 10
```

For...Next Loop

The **For** statement executes a group of statements a specific number of times. It uses a counter variable that is incremented (or decremented) by a step value with each loop, until an end limit is reached. The syntax is as follows:

For *CountVar = StartExp* **To** *EndExp* [**Step** *StepExp*]

Statement(s)

Next [*CountVar*]

When the **For** statement is first entered, *CountVar* is set to *StartExp* and then compared with the value of *EndExp.*

If *CountVar <= EndExp* then the *Statement(s)* are executed. If *CountVar > EndExp* then the *Statement(s)* are not executed. Execution continues with the first statement after the **Next** statement.

If the *Statement(s)* are executed then on reaching **Next** control returns to the start of the **For** statement. *CountVar* is then incremented (or decremented) by *StepExp.*

Note: *If StepExp is negative then StartExp must be greater than or equal to EndExp.*

After incrementing *CountVar*, the same comparison of *CountVar* and *EndExp* is then done and the loop is conditionally executed.

The counter variable following the **Next** keyword is optional and if omitted the counter variable of the last encountered **For** expression is used.

The following simple example shows the use of a negative step variable:

```
For i = 100 To 1 Step -1
     Display (Str$ (i))
Next
```

This will display the values 100, 99 .. 1. On exit *i* = 0.

GoTo Statement

With the **GoTo** statement the execution of the script is transferred to a specified line within the same procedure. The syntax is as follows:

GoTo *Label*

Label is the line label for the transfer of execution. This label **must** appear in the procedure containing the **GoTo** statement. **GoTo** statements can make it difficult to maintain and debug program code, so where ever possible use one of the other control structures.

```
If Time = 12 Then
     GoTo Noon
ElseIf Time = 24 Then
     GoTo Midnight
…
…
Noon:
…

Midnight
…
```

In the above example, if the *Time* variable is equal to12 then script execution transfers to the line marked by the label, *Noon*. If the *Time* variable is equal to 24 then script execution transfers to the line marked by the label, *Midnight*.

If Statements

TTWIN BASIC supports two forms of the **If** statement. A single line, **If...Then...Else** and a multi-line, **If...Then...ElseIf** form. **If** statements allows the excution of statements based on a condition.

In the single-line form, the statement, including those statements conditionally executed, must be one line. The syntax is as follows:

If *Expression* **Then** *Statement1(s)* [**Else** *Statement2(s)*]

If the value of *Expression* is non zero, it is treated as TRUE and *Statement1(s)* is executed. If the value of *Expression* is zero, then *Statement2(s)*, if it exists, is executed.

For example,

```
If Number >= 0 Then Display ("Positive") Else Display ("Negative")
```

This will display either “Positive” or “Negative” depending on the sign of *Number*.

The mutli-line form is particularly useful when several different outcomes are possible with each outcome requiring a different action. It also provides a structure equivalent to the **Select Case** statement. The syntax is:

If *Expression* **Then**

Statement(s)

[**ElseIf** *ElseIfExpression* **Then**

ElseIfStatement(s)]

[**Else**

ElseStatement(s)]

End If

Note: *Unlike the **If...Then...Else** the **If...Then...ElseIf** must end with an **End If**.*

If *Expression* is TRUE then *Statement(s)* are executed. If *Expression* is FALSE then the first *ElseIfExpression*, if it exists, is evaluated.

Each *ElseIfExpression* (if any) is evaluted in turn and if the *ElseIfExpression* is TRUE then the *ElseIfStatement(s)* following it is executed. If ALL of the *ElseIfExpressions* are FALSE then the *ElseStatement(s)*, if any, are executed.

For example,

```
If Month = 1 Then
     Display "January"
ElseIf  Month = 2 Then
     Display "February"
ElseIf  Month = 3 Then
     ...

Else
     Display "December"
End If
```

With this example a string is displayed depending the value of the integer *Month*.

Select Case Statement

The **Select Case** statement executes one of several groups of statements, based on one or more conditions. The **Select Case** statement is often used in situations where one or more conditions require the same action. The syntax is,

Select Case *TestExp*

[**Case** *ConditionList*

[*Statement(s)*]] ...

[**Case Else**

[*ElseStatement(s)*]]

End Select

where *TestExp*, a numeric or string expression, is the test condition and *ConditionList* is a list of conditions to be matched. *ConditionList* can be any or all of the following:

- a **To** expression, e.g, **Case** 5 **To** 7
- an **Is** comparison, e.g., **Case** *ItemA* **Is** *ItemB*
- a group of expressions each seperated by comma, e.g., **Case** 6, 7, 8 **To** 11
- a comparison e.g., **Case Is** > 3.

There can be as many **Case** *ConditionList* statements as outcomes, with each *ConditionList* evaluted in turn. If *TestExp* matches any item in the *ConditionList* then the *Statement(s)* following it are executed.

If none of the items in any of the **Case** *ConditionList* statements are matched then the *ElseStatement(s)*, if any, are executed.

You can have multiple expressions or ranges in each **Case** statement. So, for example, the following:

```
Case 1 To 4, 7 To 9, 11, 13, Is > Biggest
```

You also can specify ranges and multiple expressions for character strings.

Select Case statements can be nested with each having a matching **End Select** statement.

For example,

```
User =  “Sue”
Select Case User
Case “Amanda” To “Ros”
     ...
Case <> “Sue”
     ...
Case Else
     ...
End Select
```

While Loop

The While statement executes a group of statements in a loop as long as a condition remains true. Where required, a **While** statement can acheive the same results as a **Do Loop**. The syntax is as follows:

While *Expression*

Statement(s)

Wend

Expression is evaluated each time the **While** loop is executed. If *Expression* is TRUE then S*tatement(s)* is executed. If *Expression* is FALSE, the next statement after the **Wend** statement is executed.

For example,

```
Person = "John"
Length = Len (Person)
While Length
     Display Person
     Length = Length - 1
     Person = Left (Person, Length)
Wend
```

This loop will print the words "John", "Joh", "Jo", "J" on one line, without spaces .

Chapter 4
Procedures

Procedures are central to programming in TTWIN BASIC. They are the building blocks that allow you to structure your programs into discrete logical sections.

There are two types of procedures:

- Functions - procedures that return a value,
  and
- Subroutines - procedures that do not return a value.

Built-in Functions and Subroutines

TTWIN BASIC has over 150 built-in functions and subroutines. Some of these are implementations of Microsoft Basic others are unique to TTWIN BASIC. (Refer to *Part Two: Language Reference* for a complete description of the built-in procedures.)

In some instances there are two versions of the one function: one that returns a **String** and one that returns a **Variant**.

The **String** version uses less memory and is therefore more efficient. On the other hand, the **Variant** version allows automatic conversion between different types of data. The **Variant** version also allows **Null** values to be propagated through an expression.

The **String** version is indicated by a dollar sign (**$**) appended to the function name. The syntax is the same for both versions. The following functions have both a **String** and a **Variant** version:

† - *indicates that the keyword, function, statement or subroutine is an implementation of Visual Basic.*

†Chr$
DDERequest$
†Environ$
†Error$
GetIniFile$
GetSessionFile$
GetStatusMsg$
GetTitle$
GetTTWinPath$
†Hex$
†Input$
†LCase$
†Left$
†LTrim$
†Mid$
†Oct$
ReadHost$
ReadHostLine$
Replace$
†Right$
†RTrim$
†Space$
†Str$
†String$
†Trim$
†UCase$

Note: ***GetDDEItem$*** and ***ReadHostLine$*** have ***only*** a string form.

Calling a Function or Subroutine

A function is called by using the function name in an expression or assigning it to a variable. All of the following statements will call the function *SampleFunct*:

```
Result = SampleFunct1(Param)
If SampleFunct1(Param)< Max Then ...
Display SampleFunct1(Param) + Chr$(13)
```

You can also call a function as follows, but in both cases the **return value is discarded**:

```
Call SampleFunct2(Param)
SampleFunct2 Param
```

Unlike a function, a subroutine does not return a value in its name and so cannot be called by using it in an expression or by assigning it to a variable. A subroutine is called by a stand-alone statement.

There are two ways to call a subroutine:

```
Call SampleSub (Param1, Param2)
```

OR

```
SampleSub Param1, Param2
```


User Defined Functions and Subroutines

The user defined function and subroutine facility within TTWIN BASIC has been made to conform as closely as possible with the Microsoft Visual Basic implementation. An outline of the implemented features follows.

The syntax for defining a function or subroutine is the same as Visual Basic.

[**Static**]**Function** *FunctName* ([**ByVal**] *Parameters* [**As** *Type*])[**As** *Type*]**...**)

Statements

End Function

and

[**Static**]**Sub** *SubRoutName* ([**ByVal**] *Parameters* [**As** *Type*]**...**)

Statements

End Sub

Some examples are shown below.

```
Function Square (Value As Integer) As Integer
     Square = Value * Value
End Function
Sub Msg (MsgStr As String, Heading As String)
     MsgBOx MsgStr, Heading
EndSub
```

Defining Parameters

Parameters are optional to all user defined functions and subroutines. A single parameter is defined as

ParamName **As** *Type*

where *ParamName* is a user defined name for the parameter, **As** is a keyword and *Type* is one of the standard TTWIN BASIC types. (Refer to *Appendix A Data Type Definitions on page 377* for information on data types.)

Multiple parameter definitions are separated by commas.

For example

```
Sub StrChange (ParamName As String, Count As Integer)
```

Passing Variables by Reference or Value

The parameter declarations shown above will result in all parameters being passed by reference. Hence changing a parameter variable inside the subroutine body will also result in a change of the external variable that was passed. Preceding an individual parameter declaration with the keyword **ByVal** will result in it being passed by the value rather than by reference. If you fail to specify either the **ByRef** or the **ByVal** keywords, variables are by default passed by reference.

For example,

```
Sub Do (ByVal VarName As Integer)
```

In this case, changing the parameter inside the subroutine will not affect the external variable that was passed as a parameter.

Local Variables

All variables defined within subroutines and functions are temporary variables. This applies whether they are defined explicitly (i.e., with a **Dim** statement) or implicitly (automatic variable creation).

A temporary variable only exists while that procedure is active and hence its value is not retained between calls. If the procedure calls itself (recursion) then there will normally be two separate instances of each local variable.

If, however, a local variable is made **Static** it retains its value between calls and there is only ever one instance of the variable. This can be done in one of two ways. Either an individual variable can be declared as static or the entire procedure can be, in which case all variables within the procedure become static.

```
Sub MySub
     Dim Count As Integer
     Static LastValue As Integer
     .
     .
```

In the above example, *Count* is temporary and *LastValue* is static.

```
Static Sub MySub
     Dim Count As Integer
     Static LastValue As Integer
```

In this example both variables are static since the subroutine has been declared as static.

Exiting a Function or Subroutine

A script will automatically return from a function or subroutine when it reaches the **End Sub** or **End Function** statement at the end.

Additionally, **Exit Sub** and **Exit Function** statements can be used within the bodies of subroutines and functions respectively. Execution will immediately return from the procedure upon reaching one of these statements.

GoTo Statements within Procedures

GoTo statements are permitted within procedures, but it is illegal to attempt to jump outside the bounds of the routine. **GoTo** labels within procedures must still be unique across the entire script.

Forward Declarations

Forward declarations follow the Visual Basic standard, for example,

```
Declare Sub Msg (MsgStr As String, Heading As String)
Declare Function Square (Value AS Integer) As Integer
```

The Main Subroutine

Note: *This aspect of scripting is unique to TTWIN BASIC.*

If a subroutine called **Main** is provided then execution will start with the **Main** subroutine. If one is not provided then execution starts at the first line of code that is not part of a subroutine or function.

The **Main** subroutine **must** be defined as a subroutine with no parameters. A compile error will result if it has parameters or is defined as a function.

The following examples illustrate this point. These examples are for a DEC VT220 emulation.

Example 1:

```
Option Explicit
Static BoxOn$ As String
Static BoxOff$ As String
```

```
Static Eol$ As String
Static CrtlO$ As String
Static CrtlN$ As String

Sub Init
     CrtlN$ = Chr$(14)'SO
     CrtlO$ = Chr$(15)'SI
     Eol$ = Chr$(10) + Chr$(13)                    'LF and CR
     BoxOn$ = Chr$(27) + "(0"          'DEC VT220 line drawing on
     BoxOff$ = Chr$(27) + "(B"         'DEC VT220 line drawing off
End Sub

Init                                    ' Execution starts here
Display "Below is a box" + Eol$
Display BoxOn$
Display CrtlN$ + Eol$
Display "          lqqqqqqk" + Eol$
Display "          x      x" + Eol$
Display "          mqqqqqqj" + Eol
Display Eol$ + CrtlO$
Dispaly BoxOff$
```

In the above example a simple box is drawn, *Figure 4.1*.

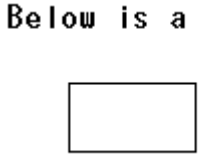


Figure 4.1: Example 1

Example 2:

Note: *The code that occurs before the* ***Main*** *subroutine is never reached.*

```
Option Explicit
Static BoxOn$ As String
Static BoxOff$ As String
Static Eol$ As String
Static CrtlO$ As String
Static CrtlN$ As String
CrtlN$ = Chr$(14)'SO
CrtlO$ = Chr$(15)'SI
Eol$ = Chr$(10) + Chr$(13)                              'LF and CR
     BoxOn$ = Chr$(27) + ")0"           'DEC VT220 line drawing on
     BoxOff$ = Chr$(27) + "(B"         'DEC VT220 line drawing off

Sub Main                                      'Execution starts here
     Display "Below is a box" + Eol$
     Display BoxOn$
     Display CrtlN$ + Eol$
     Display "          lqqqqqqk" + Eol$
     Display "          x      x" + Eol$
     Display "          mqqqqqqj" + Eol
     Display Eol$ + CrtlO$
     Dispaly BoxOff$
End Sub
```

In the above example, because the assignment statements before the main subroutine are never reached, special line drawing characters are not displayed, see *Figure 4.2*.

```
Below is a box          lqqqqqqk          x      x          mqqqqqqj
```

Figure 4.2: Example 2

Calling Routines in External DLLs

The following is a brief description of this facility. TTWIN BASIC conforms to the Visual Basic implementation. The *Microsoft Visual Basic Programmers Guide* describes this feature in detail.

Calling procedures in other DLLs allow the programmer access to existing routines without having to rewrite them or to have them present when the program is compiled. To activate a procedure in a DLL it must first be declared so that the calling program knows where to look for it.

The information in the declare statement is expressed on one line and takes the following forms:

Declare Sub *ProcedureName* **Lib** "*lib.dll*" [**Alias** "*AliasName*"] ([**ByVal**] *VarName* [**As** *Type*]**...**)

Statements

End Sub

Declare Function *ProcedureName* **Lib** *Library$* [**Alias** *AliasName*$]([**ByVal**] *VarName* [**As** *Type*])[**As** *Type*]**...**)

Statements

End Function

Where *Library$* is the path name for the dynamic-link library, `.DLL` file.

Alias indicates that the procedure has another name in the DLL. *AliasName$* is the name used in the DLL.

Use **Sub** to declare the procedure if it does not return a value. Once the procedure is declared you can call it just as you would call any user defined procedure with parameters.

Passing Parameters to DLL Procedures

Just as with user defined procedures all parameters are passed by reference which means that the address of the variable is passed. Some DLL procedures expect a parameter passed by value i.e., they expect an actual value not a memory location To pass a variable by value include the **ByVal** keyword in the declaration.

Note: *C expects all parameters to be passed by value except arrays.*

Parameters passed to external DLLs can be declared by the following types:

Parameter	Description
ByVal *Param* **As Integer**	The integer variable is passed by value rather than by reference.
ByVal *Param* **As Long**	The variable is passed by value rather than by reference. (In C language syntax - BOOL, DWORD, INT, LONG, UINT, WORD.)
ByVal *Param* **As String**	Reference to a string. (This is a BSTR string that looks like a C language LPSTR string.) This is different from passing parameters to internal procedures (Refer to *Passing Variables by Reference or Value on page 32*) ***Note: ByVal** on a string variable does not pass the value. **ByVal** ensures the string is null-terminated and the consequent pass is by reference.* Passing strings requires the user to be aware that the string should be capable of containing what the procedure returns. An overlong returned string will cause a a memory problem. As a precaution, pad the string with the maximum number of characters before calling the procedure. For example, String$ (100, 255) creates a blank string of 100 characters.

Parameter	Description
Param **As Any**	To pass a parameter to an external DLL that allows more than one data type. **As Any** removes the restriction on data type and assumes that the parameter is passed by reference. To pass a parameter by value include **ByVal** in the actual procedure call.
Param **As Integer**	Reference to an integer. (In C language syntax - LPINT.)
Param **As Long**	Reference to a long integer. (In C language syntax - LPDWORD.)
user defined types	You can pass elements of a user defined data type or the entire user defined type. User defined types can only be passed by reference.

Note: *The documentation on the DLL should tell you what sort of variable(s) it can handle. TTWIN's scripting language* ***WILL NOT*** *verify the pass and an incorrect variable may cause a failure.*

Example:

```
Declare Function DoThis Lib "lib1.dll" (ByVal AZ As Integer) As Long
...
...
Sub
     Result = DoThis(AZ)
     If Result = ....
     ....
End Sub
```

Chapter 5
Dialog Boxes

Dialog boxes are an effective way of displaying information and prompting for user input. TTWIN BASIC has two types of dialog boxes:

- prefined built-in dialog boxes that are supplied with TTWIN BASIC
- custom dialog boxes that are defined by the user.

Built-in Dialog boxes

TTWIN BASIC comes with a built-in standard dialog box, **MsgBox**, that has both a subroutine and a function form. The **MsgBox** is best utilized to display warnings, errors and brief messages where a limited response is required of the user. The user **must** respond for the application to continue.

MsgBox has 4 definable components:

- Title
- Message
- Buttons - Abort, Cancel, Ignore, No, OK, Retry and Yes buttons,
- Icons - Attention, Information, Question and Stop icons.

When the function form of the **MsgBox** procedure is used, the return value indicates which button was pressed.

The syntax for the **MsgBox** procedures is as follows:

MsgBox *Message$* [, *Type%*] [, *Title$*]

Integer MsgBox (*Message$* [, *Type%*] [, *Title$*)

where *Message$* is the message that appears in the dialog box.

Type% specifies the icons and buttons to be displayed. *Type%* is the sum of values from the following groups, (the default is 0):

Style	Value	Item displayed
Buttons	0	OK button (default)
	1	OK and Cancel buttons
	2	Abort, Retry and Ignore buttons
	3	Yes, No and Cancel buttons
	4	Yes and No buttons
	5	Retry and Cancel buttons
Icons	0	No icon (default)
	16	Stop icon
	32	Question icon
	48	Attention icon
	64	Information icon
Button action	0	First button is the default
	256	Second button is the default
	512	Third button is the default

Title$* is the title of the dialog box. If *Title$ is omitted, the title is inherited from the main TTWIN window.

Note: *As the **MsgBox** subroutine does not return a value, the use of buttons other than the **OK** button is not recommended. If you wish to utilize other buttons, use the **MsgBox** function.*

The following example is a simple illustration of the **Msgbox** subroutine,

```
Option Explicit
Dim Message As String
Dim Title As String

Message = "This application will be aborted."
Title = "FATAL ERROR."
MsgBox Message, Title
```

The box displayed by the above example is as follows:

Figure 5.1: Using the built-in **MsgBox** dialog box

As no button or icon was specified, by default only the OK button is displayed. The dialog boxes closes when the user clicks on the OK button.

Creating a Custom Dialog Box

If you require complex user input then you will need to design your own dialog box. There are 3 steps for creating a custom dialog box, as follows:

1. Defining a dialog box type using a **Begin Dialog...End Dialog** statement. See *Defining a Dialog Box Type on page 44*.
2. Declaring a variable as a dialog box with a **Dim** statement. See *Declaring a Variable as a Dialog Box on page 47*.
3. Displaying a custom dialog box by way of the **Dialog** statement. See *Displaying a Custom Dialog Box on page 48*.

Defining a Dialog Box Type

Note: *Any variables referenced in the dialog box type definition* ***MUST*** *be initialised before the definition.*

A dialog box type is defined with a **Begin Dialog...End Dialog** statement which has the following syntax:

Begin Dialog *DialogName* [*x, y,*] *cx, cy*

DialogDefinitionStatements

End Dialog

where *DialogName* is the name of the dialog box type,

x, y are the coordinates of the top left corner of dialog box, relative to current TTWIN window. If *x* and *y* are not specified then a value of zero is assumed which places the dialog box in the top left corner of the TTWIN window.

cx, cy are the width (*cx*) and height (*cy*) of the dialog box,

and the *DialogDefinitionStatements* can be any number of the following functions or subroutines:

Procedure	Description
Button	Defines a custom button that is labelled with user specified text. A **Button** statement **must** be preceded by a **ButtonGroup** statement. See *Adding Buttons to a Dialog Box on page 50*.
ButtonGroup	Finds which button was pressed to close a dialog box. The **ButtonGroup** statement **must** appear before the first **Button** statement. See *Finding the Button Response on page 51*.
CancelButton	Defines a Cancel button. See *Adding Buttons to a Dialog Box on page 50*.
Caption	Specifies the title of the dialog box. See *Giving a Dialog Box a Title on page 49*.
CheckBox	Defines a check box. A check box allows the user to specify an option. See *Other Dialog Box Elements on page 53*.
ComboBox	Defines a combo box. A combo displays only the first of a list of possible options. The remaining options, displayed in the order in which they are specified, can be viewed from a drop down menu. Only one option can be selected. See *Other Dialog Box Elements on page 53*.
GroupBox	Defines a group box. A group box brackets related items within an area on the dialog box. The box drawn can have a title, if required. Within a group box can be a combination of buttons, check boxes, combo boxes, list boxes, option buttons, text and text boxes. See *Grouping Elements on a Dialog Box on page 60*.
ListBox	Defines a list box. A list box displays a list of options in the order in which they are specified. If the space allocated for the list is too small then a scroll list is created. Only one option can be selected. See *Other Dialog Box Elements on page 53*.

Procedure	Description
OKButton	Defines an OK button. See *Adding Buttons to a Dialog Box on page 50*.
OptionButton	(Radio button) Defines an option button. Only one option in a group of options can be selected at any one time. See *Other Dialog Box Elements on page 53*.
OptionGroup	Defines the start of a new group of option buttons. An **OptionGroup** statement **MUST** appear before an **OptionButton** statement or group of **OptionGroup** statements. When one option button in a group is selected all others in the same group are deselected. There can be more than one **OptionGroup** statement in a dialog box. See *Other Dialog Box Elements on page 53*.
Text	Defines a line of text on a dialog box. See *Adding Text To a Dialog Box on page 57*.
TextBox	Defines a text box for user input and input echo. See *Using Input Fields on page 58*.

Note: *As clicking on a button closes the dialog box, a dialog box **MUST** have at least one button defined.*

For example, the following defines a dialog box type named *ContinueBox*.

```
Option Explicit
Dim Response As Integer
Dim Prompt As String
Prompt = "Do you wish to continue?"
Begin Dialog ContinueBox 220, 140, 140, 90
     Caption "No data available"
     GroupBox 10, 10, 120, 35,""
     Text 25, 25, 100, 10, Prompt
     OKButton 10, 53, 36, 24
     CancelButton 93, 53, 36, 24
     ButtonGroup Response
End Dialog
```

To see the dialog box created by this definition, see *Figure 5.2 on page 49*.

Declaring a Variable as a Dialog Box

When you have defined your custom dialog box type using the **Begin Dialog...End Dialog** statement, you can declare variables of that type with the **Dim** statement as follows:

Dim *DialogVar* **As** *DialogBox*

where *DialogVar* is the name of the dialog box variable and *DialogBox* is the dialog box type previously defined with the **Begin Dialog...End Dialog** statement.

So for example,

```
Option Explicit
Dim Response As Integer
Dim Prompt As String

Prompt = "Do you wish to continue?"

Begin Dialog ContinueBox 220, 140, 140, 90
     Caption "No data available"
     GroupBox 10, 10, 120, 35,""
     Text 25, 25, 100, 10, Prompt
     OKButton 10, 53, 36, 24
     CancelButton 93, 53, 36, 24
     ButtonGroup Response
End Dialog

Dim ContinuePrompt As ContinueBox
```

defines the dialog box type, *ContinueBox* and then declares the variable, *ContinuePrompt*, of that type. To see the dialog box declared by this script, see *Figure 5.2 on page 49*.

Displaying a Custom Dialog Box

Having defined the dialog box type and declared a variable of that type, you are now ready to display the dailog box. To display a dailog box use the **Dialog** subroutine. The syntax for the **Dialog** subroutine is:

Dialog *DlgBox*

where *DlgBox* is the name of the dialog box variable.

Using the previous example,

```
Option Explicit
Dim Response As Integer
Dim Prompt As String

Prompt = "Do you wish to continue?"

Begin Dialog ContinueBox 220, 140, 140, 90
     Caption "No data available"
     GroupBox 10, 10, 120, 35,""
     Text 25, 25, 100, 10, Prompt
     OKButton 10, 53, 36, 24
     CancelButton 93, 53, 36, 24
     ButtonGroup Response
End Dialog

Dim ContinuePrompt As ContinueBox

Dialog (ContinuePrompt)
If Response = -1 Then                    'Cancel button was pressed
     ...
```

```
ElseIf Response = 0 Then                    'OK button was pressed
     ...
End If
```

In the above example, first the dialog box type *ContinueBox* is defined, then the variable *ContinuePrompt* is declared. After which the dialog box *ContinuePrompt* is displayed, see *Figure 5.2 on page 49*. Finally, the return value from the **Dailog** function is tested to determine the action to be taken.

Figure 5.2: Displaying a custom dialog box

Giving a Dialog Box a Title

To add a title to the dialog box use the **Caption** statement. The syntax is:

Caption *Title$*

where *Title$* is the text to be displayed across the top of the dialog box. To see the dialog box with a title, see *Figure 5.3 on page 52*.

Adding Buttons to a Dialog Box

A button can be either user defined or one of the built-in buttons. TTWIN BASIC provides 2 built-in buttons for customized dialog boxes: OK and Cancel. Button statements **must** be used with the **ButtonGroup** subroutine which returns the button that was selected. (Pressing a button closes the dialog box.)

Button definitions are included in the definition of the dialog box type. (See also *Defining a Dialog Box Type on page 44*.)

The syntax for the 3 button statements is as follows:

Button *x, y, cx, cy, Label$*

CancelButton *x, y, cx, cy*

OKButton *x, y, cx, cy*

where *x, y* are the coordinates of the top left corner of button, relative to dialog box being configured,

cx, cy are the width (*cx*) and height (*cy*) of the button,

Label$ (**Button** subroutine only) is the text to appear on the button face.

For example,

```
Begin Dialog SettingsBox 8, 8, 190, 100
     Caption “Font Settings”
     GroupBox 10, 10, 170, 40,""
     ComboBox 20, 25, 90, 70, “Normal Bold Italics”, Attr
     OKButton 20, 65, 36, 24
     CancelButton 130, 65, 36, 24
     ButtonGroup Response
     Button 130, 20, 36, 24, “Colours...”
End Dialog
```

The above example defines 3 buttons: OK, Cancel, and Colour. See *Figure 5.3 on page 52* for the dialog box declared by the above example.

Finding the Button Response

To find out which button was pressed use the **ButtonGroup** subroutine, which has the following syntax:

ButtonGroup *Button%*

where *Button%*, an integer variable, indicates which button was selected as follows:

Button%	**Button pressed**
-1	Cancel button
0	OK button
1	First custom button, i.e., the first **Button** statement in the dialog type definition.
2	Second custom button i.e., the second **Button** statement in the dialog type definition.
etc.	...

The **ButtonGroup** statement is included in the definition of the dialog box type (see also *Defining a Dialog Box Type on page 44*) and **MUST** be before the first **Button** statement, as follows.

```
Option Explicit
Dim Attr As Integer
Dim Response As Integer
Begin Dialog SettingsBox 8, 8, 190, 100
     Caption “Font Settings”
     GroupBox 10, 10, 170, 40,""
     ComboBox 20, 25, 90, 70, “Normal Bold Italics”, Attr
     OKButton 20, 65, 36, 24
     CancelButton 130, 65, 36, 24
     ButtonGroup Response
     Button 130, 20, 36, 24, “Colours...”
End Dialog
```

```
Dim FontDlg As SettingsBox
...
...
Dialog FontDlg
If Response = 1 Then‘Colour button
     ...
ElseIf Response = 0 Then‘OK button
     ...
Else‘Cancel button
     ...
End If
```

In the above example, the value of *Response* set by **ButtonGroup** is tested to see which button was pressed and the appropriate action taken. See *Figure 5.3*.

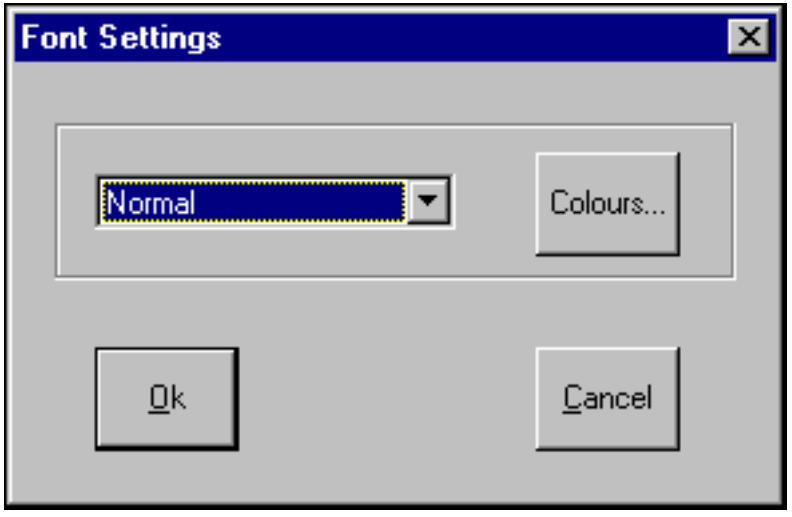


Figure 5.3: Using buttons and titles

Other Dialog Box Elements

The other dialog box elements all provide the user with a list of options. There are 4 different types of lists, as follows:

- **Check Boxes** Each option has a check box and more than one option can be selected at any one time.
- **Combo Boxes** A combo box displays **only** the first of a list of the possible options. The remaining options, in the order in which they are specified, are on a drop down menu.
- **List Boxes** A list box is a list of options in the order in which they are specified. If the space allocated for the list is too small then a scroll list is created.
- **Option Buttons** (Radio button) Each option has a button but **only** one option can be selected at any one time.

The method chosen depends on the many factors. The following is only a guideline and you must decide which is the most appropriate for your situation. If the options are independent, use check boxes. If the options in the list are mutually exclusive then use a combo box, list Box or option buttons. Use a combo box or a list box when there is a long list of options. If the space on the dialog box is limited then use a combo box.

Defining Combo Boxes and List Boxes

Combo boxes and list boxes are both useful for long lists of mutually exclusive options. For a dialog box with limited space, the combo box is more appropriate as only the first option is displayed. The other combo box options are on a drop down menu. The syntax is as follows:

ComboBox *x, y, cx, cy, Options$, State%*

ListBox *x, y, cx, cy, Options$, State%*

where *x, y* are the coordinates of the top left corner of the combo box or list box, relative to overall dialog box.

cx, cy are the width (*cx*) and height (*cy*) of the box. For the combo box, the height (*cy*) defines the combined size of the first field and the drop down menu. With a list box, if the list of options is too long for the height (*cy*)allocated then a scroll list is created.

Options$ is the list of options to be placed in the combo box or list box. This is a single string of tab separated fields. Each field is a line in the box.

State% is an integer variable identifying the current selection, as follows:

State%	**Option selected**
-1	No selection.
0	First option
1	Second option
etc.	...

Defining Check Boxes and Option Buttons

Note: *Option buttons* ***must*** *be used with an* ***OptionGroup*** *statement.*

Check boxes and option buttons (radio buttons) are more practical for short lists. With option buttons **only** one option can be selected at any one time. Where as **multiple** check boxes can be selected simultaneously. The syntax for defining check boxes and option buttons is as follows:

CheckBox *x, y, cx, cy, Option$, State%*

OptionButton *x, y, cx, cy, Option$*

where *x, y* are the coordinates of the top left corner of the Check box or Option button, relative to overall dialog box.

cx, cy are the width (*cx*) and height (*cy*) of the region, including *Option$*.

Option$ is the text to be aligned to the right of the Check box or Option button. An & preceding a character within the text will produce an ALT shortcut key. For example, "&Echo" will generate **Echo**. This enables **Echo** to be selected using the key sequence, ALT_E.

State%, (Check boxes only) an integer variable containing the control state, is set to either 1 = checked or 0 = not checked.

Option buttons do not use a state variable but instead **must** be grouped with an **OptionGroup** statement. An **OptionGroup** statement **must** appear before the first **OptionButton** statement in the group. The syntax is:

OptionGroup *Button%*

The variable *Button%* indicates which option button in the group is selected, as follows:

Button%	**Button selected**
0	First button
1	Second button
2	Third button
etc.	...

Using CheckBox, ComboBox, ListBox and OptionButton

As with buttons, check box, combo box, list box and option button definitions are incorporated in the dialog box type definition. **Remember with option buttons you MUST also have an OptionGroup statement.**

The following example illustrates all of these elements,

```
Option Explicit
Dim Response As Integer
Dim AttrState As Integer
Dim SizeState As Integer
Dim ChkState As Integer
Dim Btn As Integer
Dim Attr As String
Dim Size As String
```

```
Attrs = “Normal Bold Italics Bold&Italics”
Size = “2 4 8 12 14 18 24 32 40 48 56 64 72”

Begin Dialog TextDlg 8, 8, 240, 170
     GroupBox 20, 10, 75, 85, "Effects"
     CheckBox 24, 25, 50, 10, “&Visible”, ChkState
     OptionGroup Btn
     OptionButton  24, 40, 50, 10, “&Red”
     OptionButton  24, 55, 50, 10, “&Blue”
     OptionButton  24, 70, 50, 10, “B&lack”
     Text 120, 10, 90, 70, “Font Style”
     ComboBox 120, 20, 90, 70, Attrs, AttrState
     Text 120, 35, 90, 70, “Size”
     ListBox  120, 45, 90, 85, Size, SizeState
     OkButton       20, 130, 36, 24
     CancelButton   175, 130, 36, 24
     ButtonGroup Response
End Dialog

Dim  TextDetails as TextDlg

Dialog TextDetails
...
```

In the above example, notice that the string variables Size and Attr are assigned values **before** the dialog box type definition. *Figure 5.4* shows the dialog from the this example.

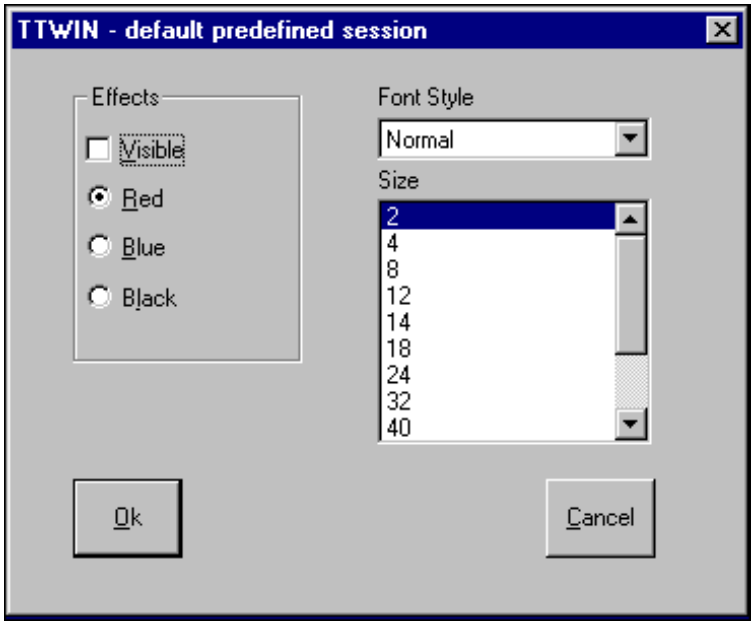


Figure 5.4: Using CheckBoxes, ComboBoxes, ListBoxes and OptionButtons

Adding Text To a Dialog Box

Text is added to a dialog box with the **Text** subroutine. This feature is most commonly used with the **Textbox** subroutine to prompt the user for input but it can be used whenever text is required, except in the title bar. (See also *Giving a Dialog Box a Title on page 49*.) The syntax is as follows,

Text *x, y, cx, cy, Text$*

where *x, y* are the coordinates of the top left corner of the text, relative to dialog box,

cx, cy are the width (*cx*) and height (*cy*) of the text,

Text$ is the text string to be placed in the dialog box.

Refer to *Using Input Fields on page 58* for an example of the **Text** subroutine.

Using Input Fields

The **TextBox** subroutine is used to defined user input fields on the dialog box. A text box **only** defines the input field. Where text is required to label the input field, use the **Text** subroutine. The syntax is as follows:

TextBox [**NoEcho**] *x, y, cx, cy, Input$*

where *x, y* are the coordinates of the top left corner of the text box, relative to dialog box,

cx, cy are the width (*cx*) and height (*cy*) of the text box,

Input$ is the string variable that contains the user input.

If **NoEcho** is specified, text appears as '*' characters, as it is typed.

For example,

```
Option Explicit
Dim UserName$ As String
Dim Password$ As String
Dim Response As Integer
Dim Failed As Integer
Begin Dialog LoginDlg 18, 18, 181, 89
     GroupBox       8, 3, 166, 40,     ""
     Text           14, 14, 41, 8,     "User name:"
     Text           14, 27, 36, 8,     "Password:"
     TextBox        58, 12, 111, 12,    UserName$
     TextBox NoEcho 58, 25, 111, 12,    Password$
     OkButton       20, 53, 36, 24
     CancelButton   120, 53, 36, 24
     ButtonGroup                        Response
End Dialog
Dim LoginDetails as LoginDlg
Dialog LoginDetails' Prompt for the login details
```

```
If Response <> 0 Then' The user pressed the cancel button
     Failed = 0
     ...
Else
     ...
          If WaitForString (10, "login: ") = 1 Then
              SendData UserName$ + Chr$(13)
              If WaitForString (10, "Password:") = 1 Then
                  SendData Password$ + Chr$(13)
                  Failed = 0
              End If
          End If
     ...
End If
If Failed Then ...
```

In the above example, see *Figure 5.5*, a simple login dialog box is displayed. The user name, which is echoed, is passed to the variable *UserName$*. The password, which is not echoed, is passed to *Password$*. The **Text** procedure is used to label both the input fields defined with **TextBox**.

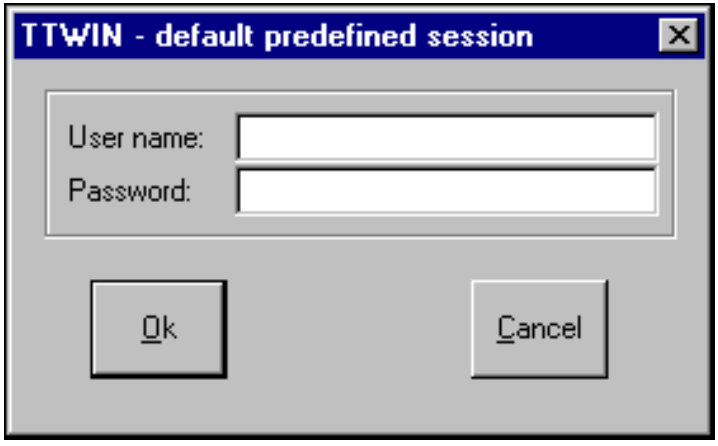


Figure 5.5: Using input fields and text

Grouping Elements on a Dialog Box

You can group items on a dialog box by drawing a box around them using the **GroupBox** subroutine. If you wish, you can also include a title to be displayed in the top left of the box. A Group box can contain any of the dialog box elements: buttons, check boxes, combo boxes, list boxes, option buttons, text and text boxes.

The syntax of the **GroupBox** subroutine is:

GroupBox *x, y, cx, cy, Title$*

where *x, y* are the coordinates of the top left corner of the Group box, relative to overall dialog box,

cx, cy are the width (*cx*) and height (*cy*) of the Group box,

Title$ is the Group box caption.

Refer to the previous example, under *Using Input Fields on page 58*.

Chapter 6
DDE Facilities

TTWIN's DDE features allow applications in a multi-tasking environment to exchange data and commands. Activities, such as cut and paste, can be automated and host applications can be linked to other applications, like Microsoft Excel, creating a streamlined and efficient working environment.

TTWIN's scripting language provides a set of predefined functions and subroutines that allow control over TTWIN's DDE facilities. This chapter contains a description of both the client and the server facilities. The operation of the actual DDE client/server is not discussed in this document.

Refer to the *Chapter 12* for a full description of the functions and subroutines mentioned in this chapter.

DDE Specific Terms

Before examining TTWIN's DDE server features it is neccessary to understand the following terms:

Term	Description
Client	The DDE client initiates a conversation with the DDE server.
Item	The actual item of data being passed. For example, for Microsoft Word an item would be a bookmark reference.

Term	Description
Server	The DDE server responds to the request for a conversation. The normal transfer of data is from the server to the client.
Service	The name of the application's service. An application can have more than one service.
Topic	The subject of the conversation. For example, for Microsoft Word a topic would be a `.DOC` file. Once a conversation has started the topic can not be changed without terminating the conversation.
Transaction	There are four types of transaction between the client and the server: • data • set data • command • advise.

DDE Server Support

The DDE server facilities allows TTWIN TO respond to a request from a DDE client. These facilities are unique to TTWIN and are not provided in this manner by other implementations.

Starting and Stopping the DDE Server

The following subroutines allow control over the naming, startup and shutting down of the DDE server in the TTWIN kernel.

Procedure	Description
SetDDEServerName	Sets the DDE service name for the current session of TTWIN.
StartDDEServer	Starts TTWIN servicing DDE requests.
StopDDEServer	Stops TTWIN servicing DDE requests.

Defining Topics and Items

Within the TTWIN kernel there are four predefined topics: **System**, **Script**, **ScreenRect** and **ScreenRegion**. You can either use these predefined topics to create new items or create new topics/item pairs.

Of the four predefined topics, **System** and **Script** have a sepcial purpose, as follows:

- **System.** Within the **System** topic is defined the item **Status** that indicates the current state of the DDE server, see *The Status Item on page 63*. You can also create new items within the **System** topic if required.
- **Script -** The **Script** topic allows the DDE client to run user defined TTWIN BASIC scripts, see *Executing Client Commands on page 64*.

The following set of functions and subroutines provide the funcitonality required to maintain a set of user defined DDE topic/item pairs within the TTWIN DDE server.

Procedure	Description
CreateDDEitem	Creates a DDE item with the given topic and item strings. The topic is automatically created if it does not exist. The initial value is an empty string.
DestroyDDEItem	Destroys a DDE item with the given topic and item strings.
GetDDEitem$	Returns the current value of the specified DDE item when the client performs a poke.
IsDDEitemDefined	Determines if an DDE item exists.
SetDDEitem	Sets the value of a previously defined DDE item.

The Status Item

Note: *This feature is not actually part of the scripting language.*

The DDE server in the TTWIN kernel automatically maintains an item **Status** within the **System** topic. It is a string defining the current DDE server status in the following form:

Status code: Status description

where *Status code* is a number representing the current status and *Status description* is text describing it. The following table lists the possible **Status** codes and descriptions.

Status code	Description
0	DDE server ready
101	Insufficient memory to complete operation
102	Could not execute. A script is already running.
103	Script execution in progress.

Executing Client Commands

Note: *This feature is not actually part of the scripting language.*

The DDE server in the TTWIN kernel provides a topic called **Script** to which DDE execute requests from the client can be sent. The execute string consists of a series of TTWIN BASIC statements which are to be passed and executed. Hence this feature allows a DDE client to create and run script statements remotely.

Typically the execute string would only consist of a few statements although there is no actual restriction placed on this. Each separate script statement **must** be enclosed within curly braces, {}, instead of being separated by an end-of-line. Consider the following two line script.

```
Connect
SendData "john"                                    'user name
```

To run this script via an execute string it would be sent as:

```
{Connect} {SendData “john“ }
```

Example Server Application

The following simple example has two sections. The first section is a TTWIN BASIC script that sets up the TTWIN server. The second section is a Visual Basic script executed from within Excel.

```
REM Create Items for DDE Server Topics

Topic1$ = "System"
Topic2$ = "ScreenRect"
Item$ = "Path"
Itemx$ = "Height"
Itemy$ = "Width"
StartDDEServer
DispClearAll

CreateDDEitem Topic1$, Item$
CreateDDEitem Topic2$, Itemx$
CreateDDEitem Topic2$, Itemy$

x$=CStr(DispHeight)
y$=CStr(DispWidth)
SetDDEitem Topic1$, Item$, GetTTWinPath
SetDDEitem Topic2$, Itemx$, x$
SetDDEitem Topic2$, Itemy$, y$
```

The TTWIN BASIC script above, sets the values for the 3 items: *Item$*, *Itemx$* and *Itemy$*. These values are read by the following Visual Basic script.

```
.....................................................
'Visual Basic Client (Run from MS Excel)
'Connect to TTWin DDE Server to access topics/items
```

```
'Note: The TTWin server script must be run before this client.

Private Sub Client()
MsgBox ("We are going to talk to TTWin DDE Server")
Q$ = Chr$(34)
Eol$ = Chr$(10) + Chr$(13)

CNo = DDEInitiate("TTWin", "System")
Header$ = "TTWin DDE Server Current Status:  "
TopicList = DDERequest(CNo, "Status")
MsgBox Header$ + TopicList(1)
DDETerminate CNo

CNo = DDEInitiate("TTWin", "System")
Header$ = "TTWin Execution Path:  "
PathList = DDERequest(CNo, "Path")
MsgBox Header$ + PathList(1)
DDETerminate CNo

CNo = DDEInitiate("TTWin", "ScreenRect")
Header$ = "TTWin Emulation Screen  " + Chr$(10) + "Row="
RectListy = DDERequest(CNo, "Height")
RectListx = DDERequest(CNo, "Width")
MsgBox Header$ + CStr(RectListy(1)) + "  " + "Column=" +
CStr(RectListx(1))
DDETerminate CNo

End Sub
```

DDE Client Support

TTWIN also provides DDE client support, allowing TTWIN to establish a conversation with a DDE server within another application.

Linking to an Application

The following functions and subroutines enable TTWIN, as the client, to establish and close an advise link to a DDE server, create and terminate a conversation with an application, send a command to a DDE server and, send data to or get data from an application.

Procedure	Description
DDEAdvise	Establishes an advise link with a DDE server.
DDEExecute	Sends a command to be executed by the DDE server.
DDEInitiate	Creates a conversation with an application, returning a DDE conversation handle.
DDEPoke	Sends data to a DDE server for the specified item.
DDERequest$	Gets data from a DDE server for the item specified.
DDETerminate	Terminates a DDE conversation.
DDETerminateAll	Terminates all currently available conversation handles.
DDEUnAdvise	Terminates an advise link with a DDE server.

The following steps outline the procedure for creating a conversation:

1. Obtain a conversation handle using **DDEInitiate**.
2. Use **DDEExecute** to send the required command to the application.
3. Use **DDEPoke** or **DDERequest$** to send or get data from the DDE server.
4. Termiante the converation with **DDETerminate**.

Example Client Application

The following simple script reads a line from the TTWIN window, see *Figure 6.1*, and passes it to an Excel worksheet. After the line is read, it is stripped of unwanted information and the resulting value is pasted into an Excel document, see *Figure 6.2*, updating the cell R7C3. The Excel document must already exist with the columns and rows already defined. To update all the sales figures for all the countries the procedure would have to repeated.

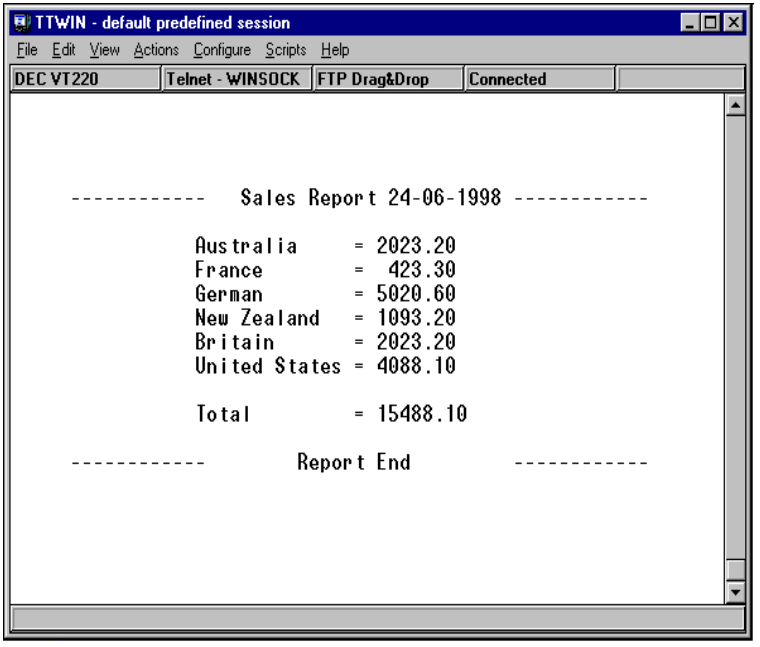


Figure 6.1: TTWIN window

```
REM Using TTWin DDE Client to establish a conversation with Excel4.0
REM TTWin Script
REM Before running this script, you must start Microsoft Excel and
REM open sales.xls
REM
```

```
Option Explicit
Dim Total$ as String
Dim Totaltmp$ as String
Dim Channel as Integer
Dim I as Integer
Dim T as Integer
Dim LENGTH as Integer
Dim A$ as String
Dim Atmp$ as String
Dim Eol$ as String

REM 'Get The Screen Data Total$

I = 0
While I < 24
     DispReadText Total$,I,10,60
     T = InStr(Total$,"New Zealand")
     if T <> 0 then
          MsgBox Total$ + " Found"
     I = 25
     End if
     I =I + 1
Wend

REM Trim the string left & right
TotalTMP$ =Ltrim(Total$)
Total$ = Rtrim(TotalTmp$)

REM Total$ is full String, Totaltmp$ is the value
T = Instr(Total$,"=")
LENGTH = Len(Total$)
```

```
Totaltmp$= Right$(Total$,(LENGTH -(T+1)))

Channel = DDEInitiate("Excel", "sales.XLS")
DDEPoke Channel, "R7C3", Totaltmp$
DDETerminate Channel
```

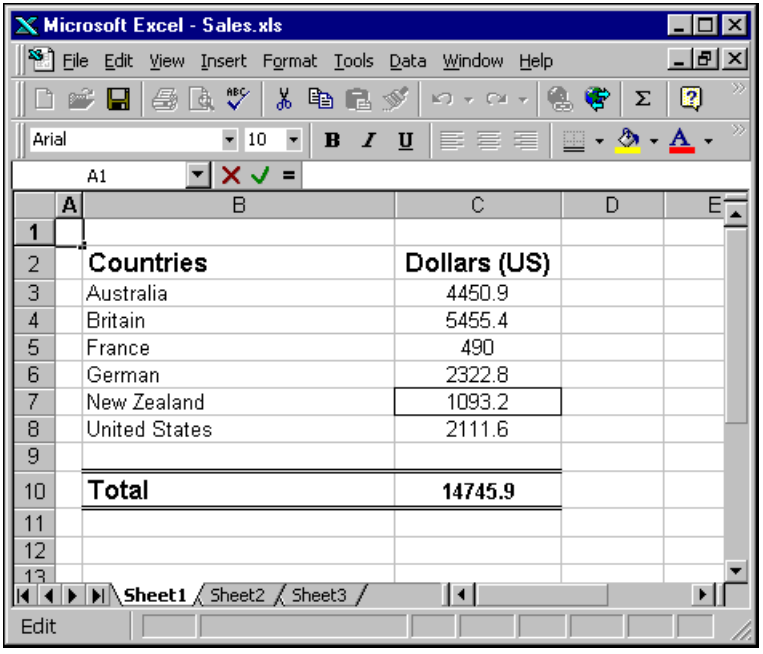


Figure 6.2: Updating cell R7C2

Chapter 7
Event Handling

In event-driven programming , a system event or a user action triggers an event procedure. The execution of the code then inevitably depends on what the user does. Event handling procedures must therefore make assumptions about a situation and your application should be structured so that these assumption are always true.

Language Elements

The following statements have been added to the grammar.

On Event *EventID* **GoTo** *ProcName* 'Start handling event

On Event *EventID* **GoTo** 0 'Stop handling event

where *ProcName* is the name of a defined subroutine or function and *EventID* can be any of the following:

HostInput	**HostOutput**
KeyPress	**MouseMove**
MouseRightDown	**MouseRightUp**
MouseLeftDown	**MouseLeftUp**
MouseMiddleDown	**MouseMiddleUp**

Event Handler Parameters and Return Types

The parameters and return type of the user procedure **must** exactly match those required for the particular event. For example, a procedure used to filter host input data would take one **String** parameter and have a return type of **String**.

Host Input and Output

Event handlers for **HostInput** and **HostOutput** events are functions returning a string with a string parameter as follows:

Function *HostEvent* (*Data* **As String**) **As String**

...

End Function

where *HostEvent* is the name of a user defined function and *Data* is the host data.

The return value of *HostEvent* is the filtered or unfiltered host data. To leave the data unfiltered, i.e., unchanged, the return value **must** be set to the input value. This is not done automatically.

In the following example, the numeric host input is unfiltered and non numeric data is converted to upper case.

```
On Event HostInput GoTo MyFilter
...
...
Function MyFilter (Data As String ) As String
     If IsNumeric(Data) Then
          Myfilter = Data
     Else
          Myfilter = UCase$(Data)
     End If
```

```
End Function
```

Mouse Events

Event handlers for all mouse events are take five integer parameters.

Sub *MouseEvent* (*X* **As Integer**, *Y* **As Integer**, *Row* **As Integer**, *Col* **As Integer**, *Shifts* **As Integer**) **As Variant**

...

End Function

where *MouseEvent* is the name of a user defined subroutine,

x and *y* are the graphics coordinates provided by the *Windows* message,

Row and *Col* are the corresponding text coordinates and

Shifts is a mask of shift states also provided by the *Windows* message. (See **WParam** of **WM_LBUTTONDOWN**).

For example,

```
On Event MouseMove GoTo MyMouseEvent
...
...
Sub MyMouseEvent (X as Integer, Y as Integer, Row as Integer, Col _
as Integer, Shifts as Integer) as Variant
     MsgBox ("You moved the mouse!!!!  Invalid response.")
End Function
```

In the above example, a message is displayed when the mouse is moved.

Keyboard Events

A keyboard event occurs when a key (a number key, letter key, punctuation key, a special key, ESC, ENTER or BACKSPACE) is pressed. An event handler for a keyboard filter has the following form:

Function *KeyEvent* (*PhysKey* **As Integer**, *EmKey* **As Integer**, *Char* **As Integer**, *Shifts* **As Integer**) **As Variant**

...

End Function

where *KeyEvent* is the name of a user defined function,

PhysKey is the key ID number from the *Windows* **WM_KEYDOWN** message (see the **VK**_... values within WINDOWS.H).

If *EmKey* > 0 then it defines an emulation specific virtual key, (not the **VK**_... values within WINDOWS.H)

If *Char* = -1 then it defines an associated ASCII character.

Shifts is a value that indicates the keboard shift states. The possible shift state values are combinations of the following:

Value	Shift state
1	Shift is pressed
2	Ctrl is pressed
4	Alt is pressed
8	Caps lock is active
16	Num lock is active
32	Scroll is active

The return value of the keyboard filter function is a **Variant** value, as follow:

Value	Meaning
Empty	A pass through unchanged.
Null	A pass through unchanged.
String	A string is used to pass a string instead of the given key. The format of the string is the same as that for the **EmKeys** subroutine. (Refer to *ttEmKeys () on page 198*.) To discard the keypress you would return an empty string (as opposed to a **Null** variant value.)
Other type	Indicates an undefined action but actually discards the key.

In the following example the key press is discarded and an empty string returned.

```
On Event KeyPress GoTo KeyDiscard
...
Function KeyDiscard (PhysKey as Integer, VirtKey as Integer, Char _
as Integer, Shifts as Integer) as Variant
     Dim EmptyString As String
     EmptyString = Empty
     KeyFilter = EmptyString
End Function
```

Restrictions within Event Handler Routines

While processing an event, it is illegal to call any proceudre that would cause the script to wait for an event. Calling any of the functions listed below during the processing of an event will result in a **fatal run time error**.

ClipboardCopy

MilliPause

DialEntry

MsgBox

DialManual

Pause

Dialog

PrintScreen

FileGet

PrintSelected

FileReceive

SendData

FileSend

WaitForString

Hangup

Chapter 8
Run-time Error Handling

Run-time errors occur during the execution of code as a result of an invalid instruction. Such an error will cause the execution of the script to be terminated unless error handling procedures are accessible. It is therefore important to incorporate run-time error handling in your script .

Below is a brief outline of the features implemented within TTWIN's scripting language to facilitate run-time error handling. The *Microsoft Visual Basic Programmer's Guide* describes this feature in detail and should be read to gain fuller understanding of its operation. TTWIN BASIC conforms to the Visual Basic description.

Enabling an Error Handler

When a run-time error occurs during the execution of a TTWIN script, the default action is to pop up an error message box and terminate the script. If this action is not appropriate then an error handler can be activated.

Enabling an error handler means indicating the section of code that execution is to jump to, in the case of an error. The **On Error** statement is used to acheive this.

It has these basic forms:

- **On Error GoTo Resume Next**

  Execution continues at the statement following the one that caused the error.
- **On Error GoTo Resume** [0] 'the 0 is optional

  Execution resumes at the statement that caused the error. That is, retry the statement.
- **On Error GoTo** *ErrorRoutine*

  When an error ocurrs, execution jumps to the error handling routine indicated by the label. (See *Resuming execution after an Error on page 78*.)
- **On Error GoTo** 0

  This statement deactivates the error handler.

Once an error handler is enabled it remains enabled while the procedure that contains it is active or until an **On Error GoTo** 0 statement.

Resuming execution after an Error

If an **On Error GoTo** *ErrorRoutine* statement is used to jump to an error handling routine, a **Resume** statment is required.

Every error handler **must** include a **Resume** statement. There are 3 basic forms:

- **Resume** [0] 'the 0 is optional

  Execution resumes at the statement that caused the error. This allows the user to enter a correction.
- **Resume Next**

  Execution resumes at the statement following the one that caused the error. In this case the error handler must correct the error.
- **Resume** *Label*

  Execution resumes at the specified *label.*

Functions and Subroutines for Error Handling

TTWIN scripting language provides the followng functions and subroutines specifically for error handling:

Procedure	Description
Assert	Generates an assertion error conditionally.
Erl	Returns the line number of an error.
Err	Returns the number of the most recent error.
Error$	Returns the description of the latest error.

For a complete description of these procedures refer to *Chapter 12 on page 127*.

```
Sub Example(Var1, Var2)
On Error GoTo ErrorHandler          'Enable error-handling routine.
...
...
Exit Sub                'No error - do not execute the error handler
ErrorHandler:                                ' Error-handling routine
     Line$ = Str$(Erl)
     ErrNo$ = Str$(Err)
     MsgBox "Line number: " + Line$ + Chr$(13) + "Error number: "_
     + ErrNo$ + Chr$(13) + "Description:" + Error$, 16, "Error"
     Select Case Err                          ' Evaluate error number
     Case -1, -2, -5
          ...
     Case Else
          ...
     End Select
     Resume Next
                ' Resume execution with the statement immediately
                       'following the line that caused the error
End Sub
```

Run-time Errors

The following is a list of TTWIN BASIC run-time errors.

Error No.	Error description
5	Illegal function call
6	Numeric overflow
9	Subscript out of range
11	Division by zero
13	Type mismatch
19	No Resume statement
20	Resume without error
52	Incorrrect file number
53	File not found
54	Incorrect file open mode
55	File already open
57	Input/output error
58	File already exists
61	Disk is full
62	Input past EOF
64	Incorrect file name
67	Too many files
68	Device is unvailable
70	Permission denied
71	Disk not ready
72	Disk media error
74	Rename across disks
75	Path or file access error
76	Path not found

Error No.	Error description
3001	Lower bound > upper bound
3002	Illegal function call in event handler
3003	Could not open capture file
3004	Assertion error
3005	Event handler already active
3006	Could not initialise DDE
3007	Could not terminate DDEconveration
3008	**DDEPoke** failed
3009	**DDERequest$** failed
3010	**DDEExecute** failed
3011	**DDEAdvise** failed
3012	Invalid DDE conversation ID
3013	**Shell** failed
3014	Only **Integer** and **Long** can be combined with **Date**
3015	Only variables of type **Integer** and **Long** can be used with **Mod**
3016	Host read failed
3017	Numeric underflow
3018	No event handlers active on **Wait**

Chapter 9
Running Scripts

Apart from running scripts from the **Scripts** menu, you can also execute a script :

- from a button
- with a hotspot
- from the keyboard
- as a startup option
- in a connect string
- initiated by the host.

In each case a macro is used as follows:

```
\S'script.tsl'
```

where `SCRIPT.TSL` is the filename of the script.

Note: *If you do not specify a path for the script, TTWIN will* ***ONLY*** *look in the directory specified in the* ***Global Preferences...*** *option on the* ***Configure*** *menu.*

See the sections below.

Executing a Script from a Button

To launch a script from a button, you need to configure a button set with a button that executes a macro string.

To do this follow these steps:

1. Select **Buttons...** from the **Configure** menu. The **Button Set Configuration** tile is then displayed.
2. Select the button set where you want to add the button. You can create a new buttons set if you wish.
3. Click on **New button...**
4. With the new button still highlighted, click on **Edit button...**. The **Edit Buttons** tile is then displayed.
5. Select **Action** from **Show attribute set**.
6. Select **String** from the options under **Assigned to...**.
7. Enter the following macro in the **Assigned value** field,

   `\S'script.tsl'`

   where `SCRIPT.TSL` is the filename of the script.
8. Click on the **OK** button (**Edit Buttons** tile) to confirm the button configuration.
9. Click on the **OK** button (**Button Set Configuration** tile) to save the button set.
10. Using the **Save** option on the **File** menu, save your `.TWC`.

Note: *You will also need to configure the other button attributes:* ***Text****,* ***Colour****,* ***Bitmap****,* ***Font*** *and* ***Alignment****. Refer to Chapter 22 - Buttons in the TTWIN User's Guide.*

For example, *Figure 9.1* shows the a button definition with the **Assigned to...** option set to **String** and the **Assigned value** field contains the macro **\S'run.tsl'**.

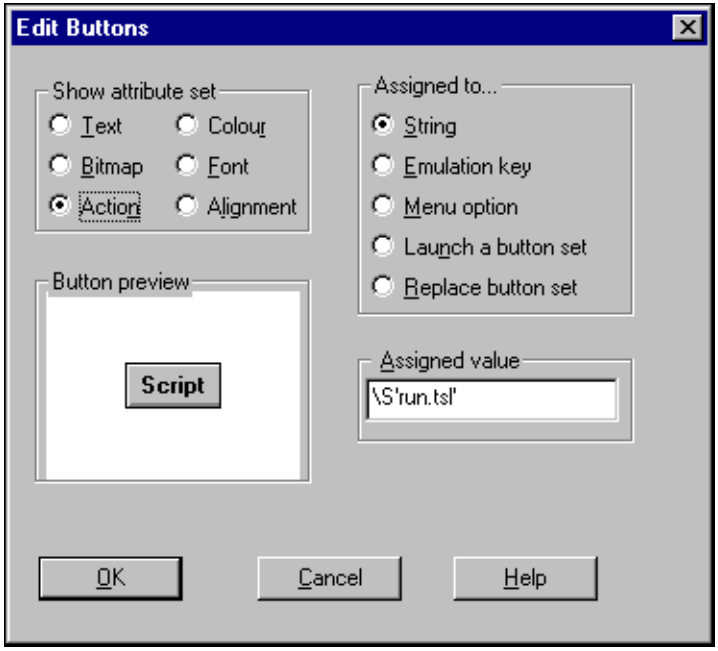


Figure 9.1: Button configuration for script execution

When this button is press the script RUN.TSL will be executed.

Executing a Script with a Hotspot

If you require a hotspot to execute a script, you will need to define a hotspot to send a string. The following steps show you how to create a new hotspot for an existing hotspot set. For further information on hotspot refer to *Chapter 24 - Hotspots in the TTWIN User's Guide.*

1. Select **HotSpots...** from the **Configure** menu. The **Configure HotSpots** tile is then displayed.
2. Click on the drop down menu on the **HotSpot Set** field and select a hotspot set. The new hotspot will be added to the set you choose.
3. Click on either the **New...** button to create a new hotspot or **Edit...** to edit an existing hotspot. The **Configure HotSpot** tile is then displayed.
4. Enter a description in the **Description** field.
5. Click on the drop down menu on the **Action Type** field and select **Send String**.
6. Click on the drop down menu on **Button Style** field and select a style for the hotspot.
7. Select the position (**Permanent** or **Relative to a Match String**) and area of the hotspot.
8. If you choose **Relative to a Match String**, click on the **String to Match...** button to specify the match string.
9. Click on the **HotSpot Action...** button. The **Configure HotSpot Action** tile is then displayed.
10. Enter the following macro in the **Send String** field,

    ```
    \S'script.tsl'
    ```

    where `SCRIPT.TSL` is the filename of the script.
11. Click on the **OK** button to confirm your hotspot action.
12. Click on the **OK** button to save your hotspot.
13. Select the **Enable HotSpots** option on the **Configure HotSpots** tile.

14. Click on the **OK** button to save the hotspot set.

15. Using the **Save** option on the **File** menu, save your `.TWC`.

Note: *You may also want to change the colour of your hotspots. To do this, click on the* ***Colours...*** *button on the the* ***Configure HotSpots*** *tile.*

For example, *Figure 9.2* shows the a hotspot configured with the macro **`\S'script1.tsl'`** entered in the **Send String** field and the **Action Type** set to **Send String**.

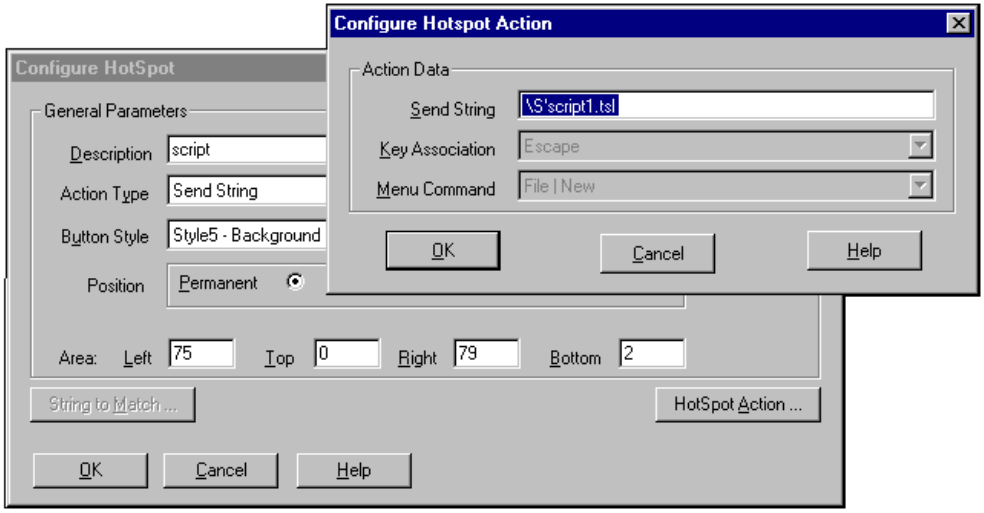


Figure 9.2: Hotspot configuration for script execution

This hotspot is displayed in the top right corner of the TTWIN window and when selected it will run the script `SCRIPT1.TSL`.

Executing a Script from a Key

If you wish to execute a script from a physical key or key sequence, you will need to map the required key(s). The following steps show you how to map a physical to execute a script:

When you are ready follow these steps:

1. Select **Keyboard...** from the **Configure** menu.
2. Check that the tile displayed is the **Keyboard Configuration - Global view** tile. If not, click on the **List by physical keys** button.
3. From the list of physical keys, highlight the PC key that you would like to assign the script.
4. Click on the **Emulation Key Edit** button. The **Emulation Key Edit** tile is then displayed.
5. Enter the following macro in the **String Assignments** column for the required shift state,

   `\S'script.tsl'`

   where `SCRIPT.TSL` is the filename of the script.
6. Click on the **OK** button to confirm your mapping.
7. Using the **Save** option on the **File** menu, save your `.TWC`.

For example, *Figure 9.3* shows the F5 key configured to launch the script `LOGIN.TSL`. The macro is **`\S'sendfile.tsl`**' entered in the **String Assignments** field for the unshifted sate of F5 key.

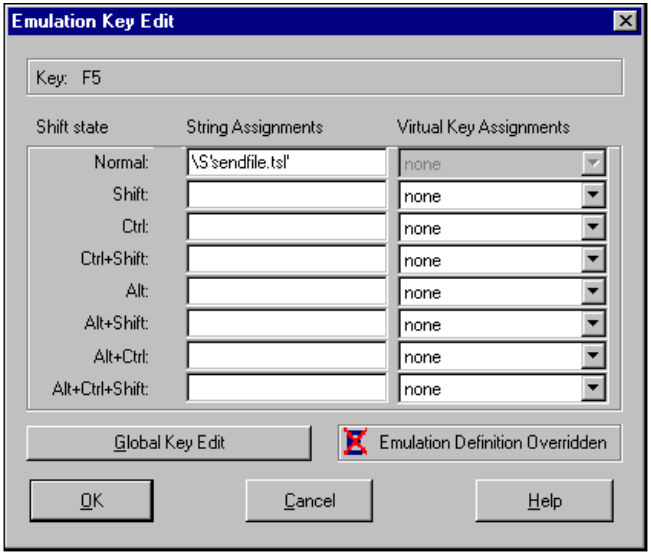


Figure 9.3: Key configuration for script execution

Executing a Script at Session Startup

There are two possibilities for executing a script when you start TTWIN. You can either use a startup string or a connect string. Startup strings run when TTWIN starts and connect strings run only after a successful connection (every connection).

Note: *Startup strings and connect strings are mutually exclusive.*

To specify a startup or connect string, follow these steps:

1. Select **Session Preferences...** from the **Configure** menu. The **Session Preferences** tile is then displayed.
2. Enter the following string in either the **Startup string** field or the **Connect string** filed ,

   `\S'script.tsl'`

   where `SCRIPT.TSL` is the filename of the script.
3. Click on the **OK** button to confirm your preferences.
4. Using the **Save** option on the **File** menu, save your `.TWC`.

For example, *Figure 9.4* shows a connect string set up in the Session Preferences,

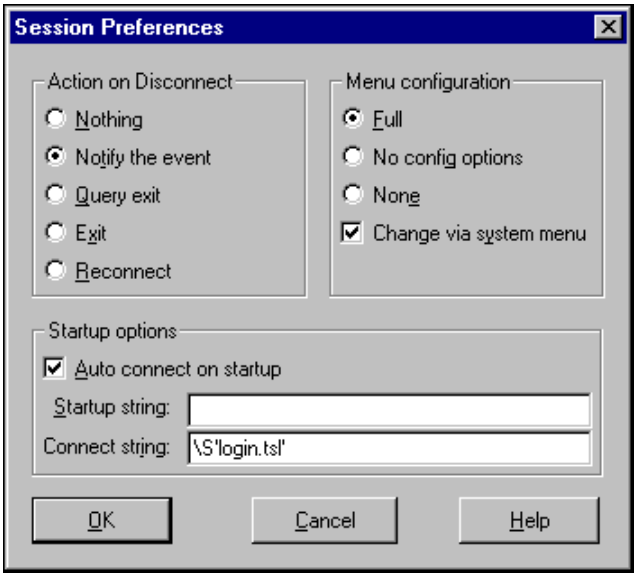


Figure 9.4: Session configuration for script execution

then when you connect (and every time you disconnect and re-connect) a script called `LOGIN.TSL` will be run. It is not necessary to give a path if the script is located in the default script directory as declared in the **Global Session Preferences**.

As **Autoconnect on startup** is also selected then TTWIN will attempt to establish a connection on startup and then run the script without user intervention.

Note: *If you are connecting via a modem you are actually going to connect to the modem, you must consider this when writing scripts for modem connections.*

Host Initiated Script Execution

Note: *This section only applies if you are able to modify applications or write programs for your host.*

It is possible to automatically trigger events in TTWIN in response to sequences sent from the host. One way to do this is to use a host initiated macro to execute a script.

The first step is to set up a macro trigger sequence in the TTWIN session configuration file (i.e., `DEFAULT.TWC`). Sending the trigger sequence from the host is dependant upon the host's operating system. You will need to be familiar with the operating system on your host. If after consulting your operating system literature you require further assistance please contact Turbosoft for guidance.

For example, in a UNIX environment you can use an ECHO command in a shell script.

Defining Host Initiated Macros

There are 2 sections in the definition of a host initiated macro:

- Trigger sequence

  The trigger sequence is defined with the **MacroSeq** profile string.

  It is recommended that you define a sequence that is unlikely to occur. For this reason ESC is not a good character to start the sequence.

  The default trigger sequence is **`\x1f~`**, where **`\x`** indicates that the next two characters are a hex value, i.e., 1F.

- Macro

  Up to 10 macros can be defined with the **Macro***N* profile string.

  Alternatively, a delimited macro that initiates a script can be sent in the host stream prefixed by *m*.

You need to modify your `.TWC` file by adding the profile strings to the [**Session**] section as follows,

[**Session**]

...

...

MacroSeq=**xxxxxxxx**

Macro*N*=**\S'script_a.tsl'**

where **xxxxxxx** is any sequence of characters. If **MacroSeq**=**xxxxxxxx** is omitted the default trigger sequence, **\x1f~**, is expected.

N, a number from 0 to 9 (inclusive), is the number of the macro to be executed.

script_A is the filename of the script. The name of the script must be enclosed with a delimiter, e.g.**'**. If you do not specify a path for the script, the script **MUST** be in the directory specified in the **Global Preferences...** option on the **Configure** menu.

The Interpretation of the Input Stream

When running the session, TTWIN scans the input stream for the trigger sequence defined with the **MacroSeq** profile string. If a trigger sequence is not defined, TTWIN scans for the default trigger sequence, **\x1f~**.

After recognising the trigger, TTWIN looks for the next character and responds as follows:

Character	Response
0	The macro defined by the profile string **Macro***0*= is executed.
...	...
9	The macro defined by the profile string **Macro***9*= is executed.
m	TTWIN continues to scan the input stream for a delimited macro. When the macro is completely read, it is played back.

For example, given the following session settings,

```
[session]
...
MacroSeq=\x1f~
Macro0=\S'zero.tsl'    - runs the script zero.tsl
Macro1=\S'first.tsl'    - runs the script first.tsl
Macro2=\S'second.tsl'    - runs the script second.tsl
Macro3=\S'third.tsl'    - runs the script third.tsl
```

If TTWIN receives **\x1f~0** from the host, **Macro0** is executed and the script ZERO.TSL runs.

If TTWIN receives **\x1f~1** from the host, **Macro1** is executed and the script FIRST.TSL runs.

And so on.

If TTWIN receives **\x1f~m** from the host, TTWIN continues to read the input stream for the macro. The input stream **\x1f~m_\S'zero.tsl'_** would produce the same result as **\x1f~0** with **Macro0** defined as above.

Chapter 10
DLL Interface

What is the TTWIN DLL Interface?

TTWIN's DLL interface represents a way of integrating PC based applications with host system legacy applications in a natural way, extending the capability of each individual system, and enhancing the look and feel of these applications. In this way the TTWIN DLL interface can greatly extend the life of the host system.

The DLL interface has the following features:

- External program control of specific TTWIN functions, normally controlled by the operator.
- Full access to the attribute and screen structure information.
- Control of different emulations without large amounts of additional code, as attribute data is provided in a standard form.
- Standard access to on screen data by the controlling program.
- Data entry into the controlling program, or creation by the controlling program as it runs, as if there was an operator entering the data through a keyboard.
- Background operation, but still providing all the functions, such as character translation, data stream management, and the interpretation of control sequences, without cluttering the screen.

Data can be easily passed to and from the emulation, interrogated and/or modified by the controlling program. In this way sophisticated new applications can be built utilising existing host based software.

The DLL Interface is also the preferred alternative to using DDE.

Using the DLL Interface

The DLL interface enables *Windows* programmers to make calls directly to TTWIN code from an external program. To use the DLL Interface you **MUST** know *Windows* programming and the use of DLL calls.

The DLL functions are defined in the library file `TWCDLL.DLL` in the `SCRIPTS` subdirectory of the `TTWIN` directory. There are also several files distributed with TTWIN to assist you. These are:

- `TWDECL.BAS` and `TWDECL32.BAS` declaration file for Visual Basic.
- `DEMO.BAS` examples of Visual Basic DLL calls and returns.
- `TWDECL.H` declaration header file for C programming.
- `TWCONST.H` declaration file for C programming.

These files are located in the `SCRIPTS` sub-directory of the `TTWIN` directory.

The DLL Interface Functions

The DLL Interface functions mirror the functions of TTWIN BASIC, but require an extra parameter for the session handle. The following table lists the functions declared in `TWDECL.BAS` and `TWDECL.H`, together with the equivalent function of TTWIN BASIC.

For general information on the parameters passed to and from the DLL functions, and the return codes, refer to the declaration files: `TWDECL.BAS`, `TWDECL32.BAS, TWDECL.H` and `TWCONST.H`.

For an overview of the DLL function refer to the description of the equivalent TTWIN BASIC function in this manual.

TTWIN BASIC	DLL
Bell	ttBell
Break	ttBreak
BtnHideSet	ttBtnHideSet
BtnShowSet	ttBtnShowSet
CfgNew	ttCfgNew
CfgOpen	ttCfgOpen
CfgSave	ttCfgSave
CfgSaveAs	ttCfgSaveAs
ClipboardCopy	ttClipboardCopy
ClipboardPaste	ttClipboardPaste
CloseHostStream	ttCloseInputCapture
Connect	ttConnect
CreateDDEItem	ttCreateDDEItem
	ttDefineVar
DestroyDDEItem	ttDestroyDDEItem
	ttDial
DialEntry	ttDialEntry
DialManual	TTDialManual
Disconnect	ttDisconnect
DispClearAll	ttDispClearAll

TTWIN BASIC	DLL
DispClearSelect	ttDispClearSelect
DispReadAttr	ttDispReadAttr
DispReadText	ttDispReadText
DispSelectAll	ttDispSelectAll
DispSelectCont	ttDispSelectCont
DispSelectRect	ttDispSelecRect
EmKeys	ttEmKeys
EmReset	ttEmReset
FileGet	ttFtGet
FileReceive	ttFtReceive
FileSend	ttFtSend
GetArgs	ttGetArgs
	ttGetCharDetail
GetCursorRow, GetCursorCol	ttGetCursorPosn
GetCursorStyle	ttGetCursorStyle
GetDDEItem	ttGetDDEItem
	ttGetFieldAttr
	ttGetFieldData
GetIniFile	ttGetIniFile
	ttGetIntVar
DispHeight, DispWidth	ttGetScreenSize
	ttGetScriptPath
	ttGetSessionByID
	(see *Writing your Application*)
	ttGetSessionByInstance
	ttGetSessionByTitle
	(see *Writing your Application*)

TTWIN BASIC	DLL
	ttGetSessionCount
	(see *Writing your Application*)
	ttGetSessionEntry
	(see *Writing your Application*)
GetSessionFile	ttGetSessionFile
	ttGetSessionList
GetStatusMsg$	ttGetStatusMsg
	ttGetStringVar
GetTitle$	ttGetTitle
GetTTWinPath	ttGetTTWinPath
Hangup	ttHangup
	ttInsertData
IsConnected	ttIsConnected
IsDDEItemDefined	ttIsDDEItem
	ttKeyText
	ttModuleCommand
	ttMoveCursor
OpenHostStream	ttOpenInputCapture
Pause	ttPause
PrintClose	ttPrintClose
PrintScreen	ttPrintScreen
PrintSelected	ttPrintSelected
ReadHost	ttReadHost
	ttReadInputCapture
	ttRegisterEventProc (C only)
	ttRunScript
	ttSearchField

TTWIN BASIC	DLL
SelectHost	ttSelectHost
	ttSelectLightPenField
SelectModem	ttSelectModem
	ttSendData
SendData	ttSendString
SetCursorStyle	ttSetCursorStyle
SetDDEItem	ttSetDDEItem
SetDDEServername	ttSetDDEServername
	ttSetHostID
	ttSetIntVar
SetMenuMode	ttSetMenuMode
SetStatusMsg	ttSetStatusMsg
	ttSetStringVar
SetTitle	ttSetTitle
SetWindowStyle	ttSetWindowStyle
ShowButtons	ttShowButtons
ShowMenu	ttShowMenu
ShowModuleLabels	ttShowModuleLabels
ShowStatus	ttShowStatus
StartCapture	ttStartCapture
StartDDEServer	ttStartDDEServer
StopCapture	ttStopCapture
StopDDEServer	ttStopDDEServer
TTWinExit	ttExit
UnlockFlow	ttUnlockFlow
WaitForString	ttWaitForString
	ttWaitForResponse (C only)

TTWIN BASIC	DLL
	ttWriteFieldString

Writing your Application

The first thing you **MUST** do (in the body of your application) is to call one or more of the DLL session access functions to get the session handle. The session access functions are:

Function	Description
ttGetSessionByID	The session ID is passed to this function to get the session handle. In order to use this function you must run `TTWIN.EXE`. with a unique session ID for each session. To do this, use the command line switch "/l [string]", where string is the unique ID. For example, **`ttwin.exe /lsession1`**.
ttGetSessionByTitle	The session title is passed to this function to get the session handle.
ttGetSessionCount	Used to establish how many session exist, if any.

Note: ttCloseSessionHandel **must** *be used to terminate the DLL conversation.*

Refer to the demonstration example, `DEMO.BAS and DEMO32.BAS`, located in the `SCRIPTS` sub-directory of the `TTWIN`.

Whether you are using Visual Basic or C you **MUST** compile the following declaration files together with your application: for Visual Basic - `TWDECL.BAS or TWDECL32.BAS` and for C - `TWDECL.H` and `TWCONST.H`.

If you wish to use TTWIN DLL calls and are experiencing difficulties please email or fax a description of the problem and examples of the calls you are making to Support.

DLL Summary

This following list tabulates all the functions that form the TTWIN DLL Interface. These elements have been grouped together based on their functionalities. For a complete description of these items, refer to *Part Two - Language Reference*

Button Sets

BtnHideSet
BtnShowSet

Communications

ttBreak
ttConnect
ttCloseInputCapture
ttDisconnect
ttInsertData
ttIsConnected
ttOpenInputCapture
ttReadInputCapture
ttSelectHost ttSendData
ttSendString
ttUnlockFlow
ttWaitForResponse
ttWaitForString

Dialling

ttDialEntry
ttDialManual
ttHangup
ttSelectModem

Emulation

EmReset

Event Handlers and Filters

ttCreateDDEItem
ttDestroyDDEItem
ttGetDDEItem
ttIsDDEItem
ttSetDDEItem
ttSetServerName
ttStartDDEServerName
ttStopDDEServerName

File Transfers

ttFtGet
ttFtReceive
ttFtSend

Miscellaneous

ttClipboardCopy
ttClipboardPaste
ttBell
ttDefineVar
ttEmKeys
ttExit
ttGetIntVar
ttGetMsgStatus
ttGetStringVar
ttGetTitle
ttKeyText
ttModuleCommand

ttPause
ttPhysicalKey
ttRunScript
ttSetHostID
ttSetIntVar
ttSetMsgStatus
ttSetStringVar
ttSetTitle
ttStartCapture
ttStopCapture
ttVirtualKey

Printing

ttPrintClose
ttPrintScreen
ttPrintSelected

Session Access

ttCloseSession
ttOpenSessionByID
ttOpenSessionByTitle
ttOpenSessionByCounter
ttGetSessionCount

Session Files

ttCfgNew
ttCfgOpen
ttCfgSave
ttCfgSaveAs
ttGetIniFile
ttGetSessionFile ttGetScriptPath

ttGetTTTWinPath

Special Emulation Interface

ttGetCharDetail
ttGetFieldAttr
ttGetFieldData
ttKeyboardLocked
ttMoveCursor
ttSearchField
ttSelectLightPen
ttWriteFieldString

Terminal Display

ttDispClearAll
ttDispClearSelect
ttDispSelectAll
ttDispSelectCont
ttDispSelectRect
ttDispReadAttr
ttDispReadText
ttGetCursorPosn
ttGetCursorStyle
ttGetScreenSize
ttSetCursorStyle

Window Manipulation

ttShowButtons
ttShowMenu
ttShowModuleLabels
ttShowStatus
ttSetMenuMode

Constants

The following table contains a list of the possible return codes for functions that form the TTWIN DLL Interface. It also includes a list of the ants and flags that are declared in the header files.

Return Codes	Value	Description
TE_OK	0	Successful operation
TE_FAIL	-1	Operation failed
TE_NOLINK	-2	No link to a TTWin session
TE_INVALIDSESSION	-3	Invalid session handle
TE_BUFFTOOSMALL	-4	Destination buffer is too small
TE_ALREADYDIALLING	-5	Cannot dial/hangup, already dialling
TE_NOMODEM	-6	Cannot dial/hangup, no modem selected
TE_ENTRYNOTFOUND	-7	Dialling/modem entry not found
TE_DIALTIMEOUT	-8	Dial operation timed out
TE_CANCELLED	-9	User cancelled the operation
TE_OUTPUTQUEUED	-10	Data was queued on a send/print
TE_NOTCONNECTED	-11	Attempted to send, but disconnected
TE_QUEUEOVERFLOW	-12	Send queue overflowed
TE_ALREADYWAITING	-13	Wait for strings when already waiting
TE_WAITTIMEOUT	-14	Wait for strings timed out
TE_ALREADYPRINTING	-15	Print request whilst already printing
TE_NOTHINGMARKED	-16	No marked region on copy/print marked
TE_UNKNOWNBSET	-17	No button set of that name

Return Codes	Value	Description
TE_ALREADYCAPTURING	-18	Capture already open
TE_UNKNOWNVAR	-19	No variable of that name

Other Flags	Value	For use with
TT_MENUFULL	0	**ttSetMenuMode**
TT_MENUNOCONFIG	1	**ttSetMenuMode**
TT_MENUNONE	2	**ttSetMenuMode**
TT_PRINTDFLT	0	**ttPrintClose**, **ttPrintScreen**, **ttPrintSelected**
TT_PRINTRAW	1	**ttPrintClose**, **ttPrintScreen**, **ttPrintSelected**
TT_PRINTWINDOWS	2	**ttPrintClose**, **ttPrintScreen**, **ttPrintSelected**
TT_CAPFILE	0	**ttStartCapture**
TT_CAPCLIPBOARD	1	**ttStartCapture**
TT_CAPOWRITE	0	**ttStartCapture**
TT_CAPAPPEND	2	**ttStartCapture**
TT_CAPRAW	0	**ttStartCapture**
TT_CAPVISUAL	4	**ttStartCapture**
TT_CURSSTEADY	80 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSBLINK	0 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSOFF	40 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSON	0 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSFULLBLOCK	1(Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSSMALLBLOCK	2 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSHORIZLINE	3 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**

Other Flags	Value	For use with
TT_CURSVERTLINE	4 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_WINSHOW	5	**ttSetWindowStyle**
TT_WINHIDE	0	**ttSetWindowStyle**
TT_WINMAXIMISE	3	**ttSetWindowStyle**
TT_WINMAXIMIZE	3	**ttSetWindowStyle**
TT_WINMINIMISE	2	**ttSetWindowStyle**
TT_WINMINIMIZE	2	**ttSetWindowStyle**
TT_WINRESTORE	9	
TT_BTNTOP	0	**ttBtnShowSet**
TT_BTNBOTTOM	1	**ttBtnShowSet**
TT_BTNLEFT	2	**ttBtnShowSet**
TT_BTNRIGHT	3	**ttBtnShowSet**
TT_BTNHORIZFLOAT	4	**ttBtnShowSet**
TT_BTNVERTFLOAT	5	**ttBtnShowSet**
TT_EMMODULE	1	
TT_CMMODULE	2	
TT_FTMODULE	3	
TT_STRINGVAR	1	
TT_INTVAR	2	

Chapter 11
TTWIN ActiveX Controls

What is TTWIN ActiveX Controls?

TTWIN ActiveX Controls are created using the Component Object Model (COM) to enable software components to interact with one another. They are run within a variety of containers in a 32-bit environment. The most common containers you may use would be Visual Basic, Internet Explorer, Delphi and Microsoft Access.

TTWIN ActiveX Controls expose dozens of methods and properties to container applications that wish to have a full control over the TTWIN terminal emulator. Yet, TTWIN ActiveX Controls is scalable so your container applications can create a number of manageable and reusable components with each employing a small number of TTWIN ActiveX Controls methods. Later you may integrate these components into bigger applications.

How to register TTWIN ActiveX Controls

For a successful TTWIN installation, you should find TTWIN ActiveX Controls as two files in `\Bin32\ttwin.ocx and \Bin32\ttwin.tlb`. As the installation program does not do an automatic ActiveX control registration, it is your duty to register TTWIN ActiveX Controls onto your registry database. Microsoft provides a utility program `Regsvr32.exe`. For

an NT environment, you should find the executable file under system32 directory. For Windows 95/98, you should find it in the system directory. For example, suppose you have installed TTWIN in a directory `c:\ttwin32`. To register TTWIN ActiveX Controls to your Windows 95 Registry, bring up an MS-DOS prompt and type in the full path name of ttwin.ocx as shown below:

```
C:\Win95\SYSTEM>regsvr32.exe
c:\ttwin32\bin32\ttwin.ocx
```

To facilitate container applications loading the essential TTWIN DLLs, you also need to enter the path, eg `c:\ttwin32\bin32`, into your `autoexec.bat` file and restart your machine.

The List of Properties and Methods

Here we give TTWIN ActiveX Controls properties and methods in Visual Basic Applications (VBA) format. Descriptions given here are brief, but you will find a full description of most methods inside the TTWIN BASIC *Language Reference section* provided that the function names are identical.

Directly Accessible Properties

AttachTtwin	**IDStr**	**SendString**
ColCursorPosn	**IniFile**	**SessionFile**
CursorStyle	**RowCursorPosn**	**SessionName**
DeAttachTtwin	**ScreenHeight**	**SessionTitle**
GetAttachInfo	**ScreenWidth**	**TTWinExePath**

Accessible Methods

DispReadAttr	**ReadInputCapture**	**StringVar**
DispReadText	**SessFile**	**Title**
FieldData	**StatusMsg**	

Property Definitions

```
ColCursorPosn As Integer

CursorStyle As Integer

IDStr As String

IniFile As String

RowCursorPosn As Integer

ScreenHeight As Integer

ScreenWidth As Integer

SendString As String

SessionFile As String

SessionID As String

SessionName As String

SessionTitle As String

TTWinExePath As String
```

Method Definitions

```
Property AttachTtwin (Kbd As Long, Scrn As Long,
         Attach As Long) As Integer

Property DeAttachTtwin () As Integer
```

```
Property DispReadAttr(Row As Integer, Col As Integer,
        Length As Integer) As String

Property DispReadText(Row As Integer, Col As Integer,
        Length As Integer) As String

Property FieldData(Row As Integer, Col As Integer,
        BuffLength As Integer) As String

Property GetAttachInfo (lpKbd, lpScrn As Long,
        lpAttach As Long) As Integer

Property ReadInputCapture(Length As Integer) As String

Property SessFile(iLength As Integer) As String

Property StatusMsg(iLength As Integer) As String

Property StringVar(VarName As String,
        iBuffLength As Integer) As String

Property Title(iLength As Integer) As String
```

The list of methods

```
Function AddStrForWaitResponse(str As String) As Integer

Function AttachTtwin (Kbd As Long, Scrn As Long,
        Attach As Long) As Integer

Function Bell ( ) As Integer

Function Break ( ) As Integer

Function BtnHideSet (setname As String) As Integer
```

```
Function BtnShowSet(SetName As String, Location As Integer)
         As Integer

Function CfgNew() As Integer

Function CfgOpen(Filename As String) As Integer

Function CfgSave() As Integer

Function CfgSaveAs(Filename As String) As Integer

Function CheckForSessionFile(SessionFile As String) As
Integer

Function ClipboardCopy() As Integer

Function ClipboardPaste() As Integer

Function CloseInputCapture() As Integer

Function CloseSession() As Integer

Function Connect() As Integer

Function DefineVar(VarName As String, VType As Integer)
         As Integer

Function DeAttachTtwin() As Integer

Function DialEntry(EntryName As String) As Integer

Function DialManual(Number As String, Description As
String) As Integer

Function Disconnect() As Integer
```

```
Function DispClearAll() As Integer

Function DispClearSelect() As Integer

Function DispSelectAll() As Integer

Function DispSelectCont(Row1 As Integer, Col1 As Integer,
         Row2 As Integer, Col2 As Integer) As Integer

Function DispSelectRect(Row1 As Integer, Col1 As Integer,
         Row2 As Integer, Col2 As Integer) As Integer

Function EmKeys(Text As String, Length As Integer)
         As Integer

Function EmReset() As Integer

Function EnableDebugMsg(bEnable As Boolean) As Boolean

Function Exit() As Integer

Function FtGet(RemoteName As String, LocalName As String)
         As Integer

Function FtReceive(LocalName As String) As Integer

Function FtSend(LocalName As String, RemoteName As String)
         As Integer

Function GetAttachInfo (lpKbd, lpScrn As Long, lpAttach As
Long) As Integer

Function GetCharDetail(Row As Integer, Col As Integer,
          FgColour As Integer, BgColour As Integer,
          Featurs As Integer) As Integer
```

```
Function GetComCfgNames(Size As Integer) As String

Function GetComCfgString(String As String, Size As Integer)
         As String

Function GetComCfgStrings(Size As Integer) As String

Function GetDisplayData(Str, Attrs, Row As Integer,
         Col As Integer, Length As Integer) As Integer

Function GetEmulCfgNames(Size As Integer) As String

Function GetEmulCfgString(String As String,
         Size As Integer) As String

Function GetEmulCfgStrings(Size As Integer) As String

Function GetFieldAttr(StartRow As Integer,
         StartCol As Integer, FieldAttr As Integer) As
         Integer

Function GetFTCfgNames(Size As Integer) As String

Function GetFTCfgString(String As String, Size As Integer)
         As String

Function GetFTCfgStrings(Size As Integer) As String

Function GetIntVar(VarName As String, ErrorFlag As Integer)
         As Integer

Function GetTtwinCfgNames(Size As Integer) As String

Function GetTtwinCfgString(String As String,
         Size As Integer) As String
```

```
Function GetTtwinCfgStrings(Size As Integer) As String

Function Hangup() As Integer

Function InsertData(Buff As String) As Integer

Function IsConnected() As Integer

Function KeyboardLocked() As Integer

Function KeyText(Text As String) As Integer

Function MoveCursor(Row As Integer, Col As Integer)
As Integer

Function OpenInputCapture() As Integer

Function OpenSessionByCounter(iCounter As Integer)
        As Integer

Function OpenSessionByID(IDStr As String) As Integer

Function OpenSessionByTitle(SessTitle As String) As Integer

Function Pause(Ticks As Integer) As Integer

Function PhysicalKey(PhysicalKey As Integer) As Integer

Function PrintClose() As Integer

Function PrintScreen(Mode As Integer) As Integer

Function PrintSelected(Mode As Integer) As Integer

Function ResetListForWaitResponse(iSpare As Integer)
        As Integer
```

```
Function RunScript(Script As String) As Integer

Function RunTTWinExe() As Integer

Function SelectHost(Host As String) As Integer

Function SelectLightPenField(Row As Integer,
         Col As Integer) As Integer

Function SelectModem(ModemName As String) As Integer

Function SendData(Buff As String, Length As Integer)
         As Integer

Function SendStr(Data As String) As Boolean

Function SetComCfgString(String As String, Size As Integer)
         As Integer

Function SetComCfgStrings(Strings As String,
         Size As Integer) As Integer

Function SetEmulCfgString(String As String,
         Size As Integer) As Integer

Function SetEmulCfgStrings(Strings As String,
         Size As Integer) As Integer

Function SetFTCfgString(String As String, Size As Integer)
         As Integer

Function SetFTCfgStrings(Strings As String,
         Size As Integer) As Integer

Function SetHostID(HostID As String) As Integer
```

```
Function SetIntVar(VarName As String, Value As Integer)
        As Integer

Function SetMenuMode(State As Integer,
        InSystemMenu As Integer) As Integer

Function SetSessionName(SessName As String) As Boolean

Function SetSessTitle(SessTitle As String) As Boolean

Function SetTtwinCfgString(String As String,
        Size As Integer) As Integer

Function SetTtwinCfgStrings(Strings As String,
        Size As Integer) As Integer

Function SetWindowStyle(Style As Integer) As Integer

Function ShowButtons(State As Integer) As Integer

Function ShowMenu(State As Integer) As Integer

Function ShowModuleLabels(State As Integer) As Integer

Function ShowStatus(State As Integer) As Integer

Function StartCapture(Filename As String, Flags As Integer)
        As Integer

Function StopCapture() As Integer

Function SwitchComm(CommName As String) As Integer

Function SwitchEmulator(EmuName As String) As Integer

Function SwitchFTransfer(FTName As String) As Integer
```

```
Function UnlockFlow() As Integer

Function VirtualKey(VirtualKey As Integer) As Integer

Function WaitForResponse(Timeout As Integer) As Integer

Function WaitForStrAsync(lpUserHandler As Long,
         Timeout As Integer) As Integer

Function WaitForStrCancel() As Integer

Function WaitForString(Buff As String, Timeout As Integer)
         As Integer

Function WriteFieldString(Value As String,
         Length As Integer) As Integer
```

How to Use the Type Library

TTWIN ActiveX Controls come with its type library file `ttwin.tlb` that is located in the Bin32 directory. If you writing a MFC based C++ application, you can use ClassWizard to add a new class that based on `ttwin.tlb`. The application can then create instances of the wrapper class and invokes the member functions of the class, just the same way your application using an ordinary class object does. The implementation of the wrapper class in general does not require modifications.

To use the type library from within a Visual Basic project, simply create a Standard EXE project. Then you select menu item **Project | References Ö**, and select reference **TTWIN ActiveX Control module** (see Figure 11.1). If you could not find it the list box, using Browse button to locate it at `Bin32\ttwin.tlb`.

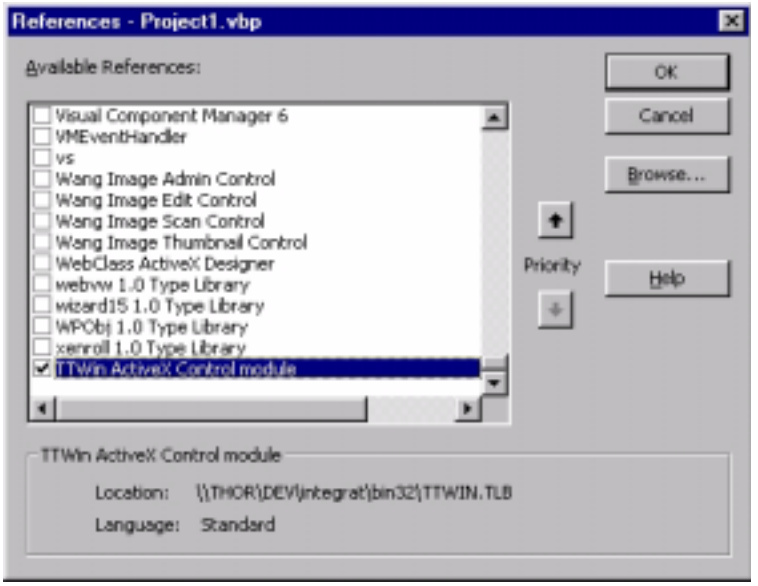


Figure 11.1: Using Type Library for a Visual Basic Project

Now add a button "**StartTtwin**" in a form and edit the code which simply creates a control object and invokes the RunTTWinExe method. This method will start a TTWIN session and is the first method you need to call.

```
Private Sub StartTtwin_Click()

Dim ttwinx As TTWLib.Ttwin

Set ttwinx  = New TTWLib.Ttwin

ttwinx.RunTTWinExe

End Sub
```

Use TTWinActiveX as a Control

You may create a Standard EXE Visual Basic project and select menu item **Project | ComponentsÖ**. Inside the Components listbox, select **TTWIN ActiveX Control module**. A yellow icon TTWIN will appear at the end of Visual Basic Toolbox. To create an instance of ttwin control, simply click on the control icon and then add the control to your form. You need to resize the control to a reasonable large square to accommodate the full TTWIN session window (see Figure 11.2).

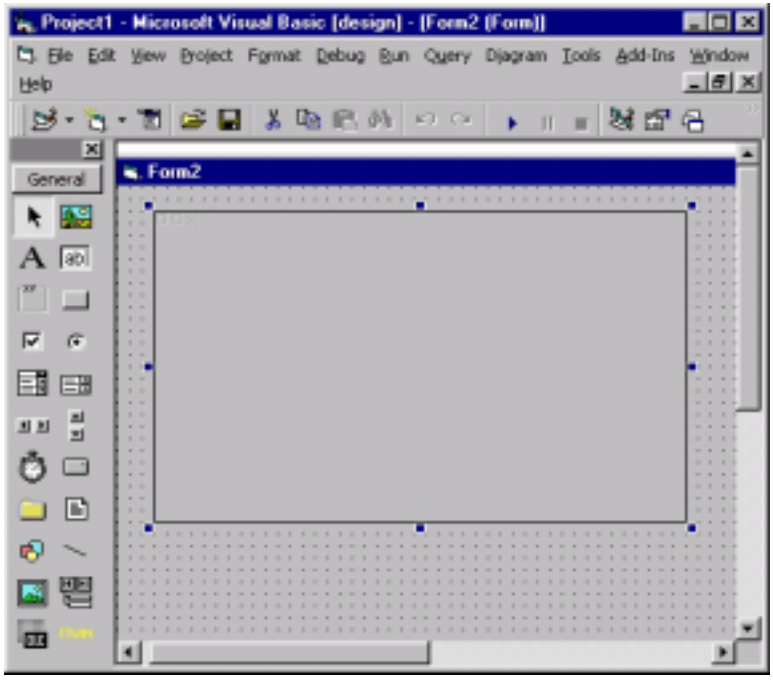


Figure 11.2: The TTWIN Control on a Visual Basic Form

In this example, we add three buttons, "**Start**", "**Connect**" and "**Disconnect**" beneath the TTWIN OCX control. The following code is a guidance to the use of TTWIN ActiveX Controls methods and properties.

Example

```
Private Sub Start_Click()
      Ttwin1.RunTTWinExe
End Sub

Private Sub Connect_Click()
      Ttwin1.Connect
End Sub
```

Figure 11.3 shows the screen snap after you press the **Start** and **Connect** buttons connectively.

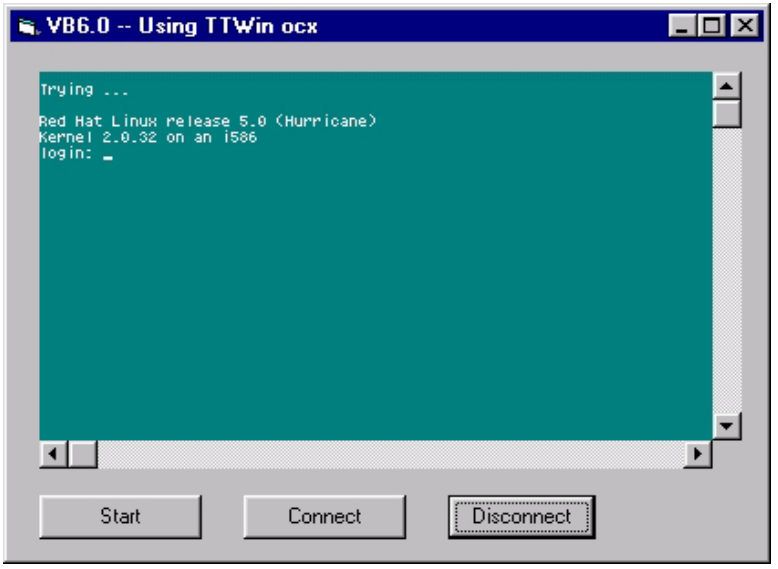
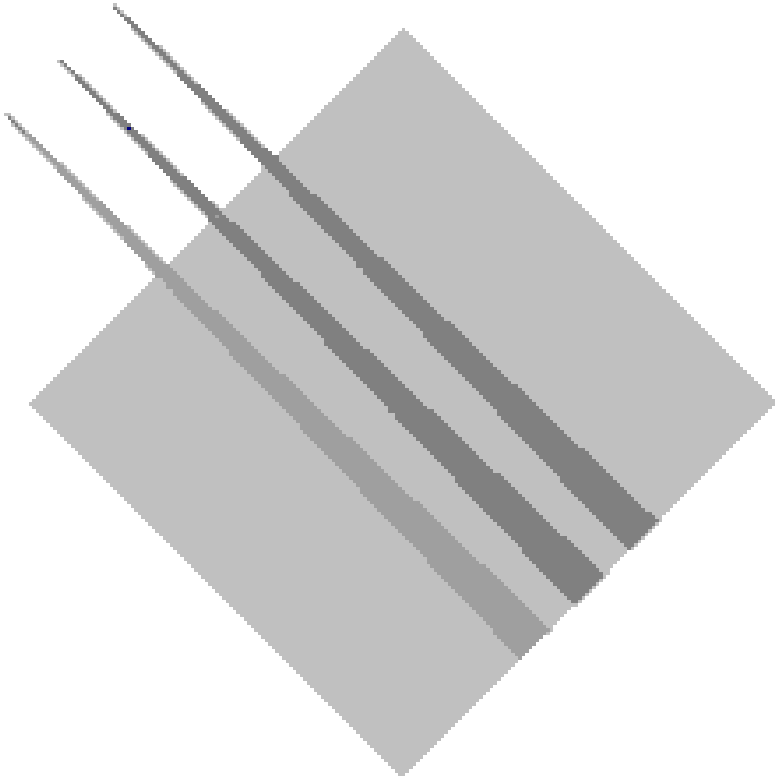


Figure 11.3: Run a TTWIN Session within a Visual Basic Application

The Limitation of TTWIN ActiveX Controls

User inputs from keyboard and mouse to TTWIN ActiveX Controls will be processed by the ttwin.exe kernel. Therefore there is no stock event notifications fired to the container applications. TTWIN ActiveX Controls come with a simple property page to allow container application designer to specify the ttwin.exe path, the session file and title. However the main aim for this property page is providing a debug mode and a release mode.

PART TWO
LANGUAGE REFERENCE



Chapter 12
Language Reference

This reference section provides a complete description of all the TTWIN BASIC functions, subroutines and control statements. If you require further information on standard Visual Basic procedures refer to the *Microsoft Visual Basic Language Reference*.

Within this section the following conventions are used:

ProcName ()	Brackets after a procedure indicate a function. If there are no parentheses after the name of a procedure then a return value is not defined. When calling a function that has no parameters the brackets are omitted.
FuncName$ ()	A dollar sign (**$**) appended to the function name indicates that the function returns a **String**. All such functions also have a **Variant** form, except GetDDEItem$ and ReadHostLine$ which has only a string form. The syntax is the same for both versions but the dollar sign (**$**) is appended to the function name **ONLY** for the **String** form.

For a complete list of the parameter notations and syntax used in this section refer to - *Conventions on page 5*.

Some of the functions pass parameters or return values that have been defined in the header files as constants or flags. Refer to the relevant header file or to *Appendix B - Constants on page 381* for further information.

' (apostrophe)

Description Indicates that the remainder of a line is a comment.

Syntax **'** This is a comment.

Example
```
x = 96      '  This is a comment
```

See also **Rem**

Abs ()

Description Returns the absolute value of the given integer parameter.

Declaration
Function Abs (**ByVal** *Num* **As Double**) **As Double**
Function Abs (**ByVal** *Num* **As Integer**) **As Integer**
Function Abs (**ByVal** *Num* **As Long**) **As Long**
Function Abs (**ByVal** *Num* **As Variant**) **As Variant**

Parameter *Num* is any valid numeric expression.

Return value Returns the absolute value of *Num*.

Example
```
Abs (-4.3)
```
Returns 4.3, the absolute value of -4.3.

AppActivate

Description Activates an application window.

Declaration **Sub AppActivate** (**ByVal** *Window$* **As String** [,**ByVal** *Delay%* **As Integer**])

Parameters *Window$* is the caption on the top of the application window.

Delay% indicates whether the specified application has immediate focus or not.

Delay%	Meaning
0	*Window$* is activated immediately.
1	The activation of *Window$* is delayed until the calling application has the focus.

Example

```
AppActivate "Microsoft Excel"
```

Asc ()

Description Returns the ASCII value for the first character of a string.

Declaration **Function Asc** (**ByVal** *Str$* **As String**) **As Integer**

Parameter *Str$* is a valid string expression.

Return value Returns the ASCII value for the first character in *Str$*.

Example `Asc ("ABCDEF")`

Returns 65, theASCII value of A.

See also **Chr$**, **Str$**

Assert

Description Generates an assertion error conditionally.

Declaration **Sub Assert** (*Flag%* **As Integer**)

Remarks **Assert** displays a message box indicating the line number of the **Assert** statement.

Parameter The value of *Flag%* determines whether an error is generated, as follows:

Flag%	Description
1	Error generated
0	Error not generated.

Example

```
If DispHeight < 5
     Assert (1)
End If
```

See also **Err**, **Erl**, **Error$**

Atn ()

Description Calculates the arctangent of a number.

Declaration **Function Double Atn** (**ByVal** *Expression* **As Double**)
Function Single Atn (**ByVal** *Expression* **As Single**)

Remarks The function returns the value in radians. (The arctangent is the inverse trigonometric function of tan.)

Parameter *Expression* is any valid numeric expression.

Return value Returns the arctangent of *Expression*.

Example
```
Radians = Atn (x)
```

See also **Cos**, **Sin**, **Tan**

Beep

Description Plays a sound.

Declaration **Sub Beep**

Remarks The pitch and volume of the sound depends on the software and hardware installed on your PC.

Example
```
Alert = 3
While Alert
     Beep
```

```
    Alert = Alert - 1
Wend
```

Three warning beeps are sounded.

See also **Bell**

Begin Dialog ... End Dialog

Description Defines a dialog.

Syntax **Begin Dialog** *DialogName* [*x*, *y*,] *cx*, *cy*
DialogDefinitionStatements
End Dialog

Remarks A dialog must have at least one **OKButton**, **CancelButton** or **Button** control defined.

Parameters *DialogName* used as a type to define variables.

x, *y* are the horizontal (*x*) and vertical (*y*) position of top left corner of the dialog box, relative to current TTWIN window. If not specified then a value of zero is assumed which places the dialog box in the top left corner.

cx, *cy* are the control width (*cx*) and control height (*cy*) of the dialog box.

DialogDefinitionStatements includes any of the following:

Button, **ButtonGroup**, **CancelButton**, **Caption**, **CheckBox**, **ComboBox**, **GroupBox**, **ListBox**, **OKButton**, **OptionButton**, **OptionGroup**, **Text**, and **TextBox**.

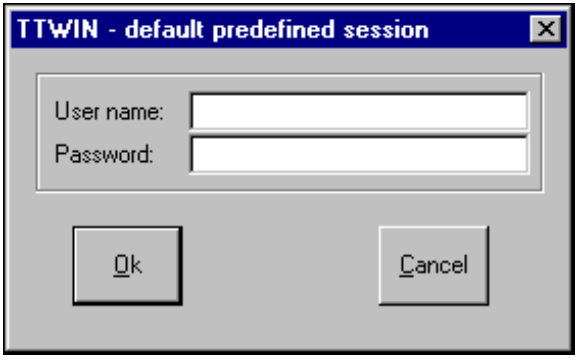
Note: *Any variables referenced by the DialogDefinitionStatements* ***MUST*** *be initialised before the definition.*

These variables are passed to the **Dialog** statement to run the dialog.

Example

```
Option Explicit
Dim UserName$ As String
Dim Password$ As String
Dim Response As Integer
Dim Failed As Integer
Begin Dialog LoginDlg 18, 18, 181, 89
     GroupBox       8, 3, 166, 40,     ""
     Text           14, 14, 41, 8,     "User name:"
     Text           14, 27, 36, 8,     "Password:"
     TextBox        58, 12, 111, 12,    UserName$
     TextBox NoEcho 58, 25, 111, 12,    Password$
     OKButton       20, 53, 36, 24
     CancelButton   120, 53, 36, 24
     ButtonGroup                        Response
End Dialog
```

After declaring a variable of type *LoginDlg* and running the dialog using the **Dailog** statement, the above dialog definition will generate the following:



See also **Dialog**

ttBell ()

Description Sounds the bell.

Declaration **Sub Bell**

Visual Basic **ttBell** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttBell** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.

Value	Meaning
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example `Bell`

Example

See also **Beep**

ttBreak ()

Description Sends a break on the communications line.

Declaration **Sub Break**

Visual Basic **ttBreak** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttBreak** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Actions > Break

Example

```
Break
```

Example

See also **Connect, Disconnect, IsConnected**

See also **ttConnect**, **ttDisconnect**, ttIsConnected

ttBtnHideSet ()

Description Hides a button set with name *SetName$*.

Declaration **Sub BtnHideSet** (**ByVal** *SetName$* **As String**)

Visual Basic

ttBtnHideSet(
ByVal *HSess* **As Long**,
ByVal *SetName* **As String**)
As Long

C language

int ttBtnHideSet (
HSess *HSess*,
LPCSTR *SetName*);

Parameter *SetName$* is the name of the button set as defined within the **Button Set Configuration** tile.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
BtnHideSet Set1
```

The button set *Set1* will be removed from view.

See also **BtnShowSet**

ttBtnShowSet

ttBtnShowSet ()

Description Shows a button set.

Declaration **Sub BtnShowSet** (**ByVal** *SetName$* **As String**, **ByVal** *Location%* **As Integer**)

Visual Basic

ttBtnShowSet(
ByVal *HSess* **As Long**,
ByVal *SetName* **As String**,
ByVal *Location* **As Long**)
As Long

C language

int ttBtnShowSet (

HSess *HSess*,
LPCSTR *SetName*,
Int *Location*);

Remarks If a button set is already in the position where the new set is to be displayed then the old set is removed to accommodate for the new set.

Parameters *SetName$* is the name of the button set as defined on the **Buttons Configuration** tile.

Location% specifies the position of the button set on the screen, as follows:

Location%	**Position**
0	Top
1	Bottom
2	Left
3	Right
4	Floating vertical
5	Floating horizontal

These values are declared as constants in the header files; `TWCONST.H` and `TWDECL.BAS`. Refer to *Appendix B - Constants on page 381* for a complete list of declared constants.

Return value The function returns the following values:

Value	**Meaning**
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
BtnShowSet  "Set1", 2
```

The button set *Set1* will be displayed down the left hand side of the TTWIN window.

See also **BtnHideSet**

ttBtnHideSet

Button

Description Defines a custom button labelled with the given text.

Syntax **Button** *x, y, cx, cy, Label$*

Remarks A **ButtonGroup** statement must appear before the first **Button** statement. Pressing a custom button closes the dialog box. The variable referred to in the **ButtonGroup** statement is set to 1 for the first custom button, 2 for the 2nd, etc.

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of button, relative to dialog box being configured.

cx, cy are the width (*cx*) and height (*cy*) of the button.

Label$ is the text to appear on the button face.

Example

```
Button  114, 53, 36, 24, "Help"
```

A button will appear in the dialog box with the word “Help” on its face as shown below.

See also **ButtonGroup**, **CancelButton**, **OKButton**

ButtonGroup

Description Finds which button was pressed to close a dialog.

Syntax **ButtonGroup** *Button%*

Parameter *Button%*, an integer variable, set to a value indicating which button was pressed. The possible settings for *Button%* are as follows:

Button%	**Button pressed**
-1	Cancel button
0	OK button
1	First custom button
2	Second custom button
etc.	...

Note: *The button number is determined by the position of the corresponding* ***Button*** *statement in the dialog type definition: 1 for the first button, 2 for the second, and so on.*

Example

```
ButtonGroup  Response
```

```
     ....
If Response = 0 Then
     .....
```

A dialog box is displayed until one of its buttons are pressed. The button that has been pressed is passed to the integer variable *Response*.

See also **Button**

Call

Description Transfers control to a function, subroutine or DLL procedure.

Syntax **Call** *ProcName* ([**ByVal**] *Parameters* [**As** *Type*]**...**)

Remarks When **Call** is used, parameters **must** be enclosed in parentheses. If you omit the **Call** keyword then you **must** omit the parentheses.

If **Call** is used to call a function, then the return value is discarded.

Parameter *ProcName* is the name of the function, subroutine or DLL procedure.

Example 1

```
Sub Run (Param1 As Integer, Param2 As Integer)
     ...
End Sub
...
Var1 = 53
```

```
Var2 = 5
...
Call Run (Var1, Var2)
```

Example 2

```
Sub Run (Param1 As Integer, Param2 As Integer)
     ...
End Sub
...
Var1 = 53
Var2 = 5
...
Run Var1, Var2
```

Both Example 1 and Example 2 give the same results. However, the **Call** keyword and the parentheses are omitted in Example 2.

CancelButton

Description Defines a cancel button.

Syntax **CancelButton** *x, y, cx, cy*

Remarks The dialog box closes when the cancel button is pressed. If there is a **ButtonGroup** statement, the specified variable will be set to -1.

Parameters *x*, *y* are the horizontal (*x*) and vertical (*y*) position of top left corner of button, relative to dialog box being configured.

cx, *cy* are the width (*cx*) and height (*cy*) of the button.

Example `CancelButton 104, 53, 36, 24`

A cancel button will appear in the dialog box as shown below.

See also **Button**, **ButtonGroup**, **OKButton**

Caption

Description Defines the title of the dialog box.

Syntax **Caption** (**ByVal** *Title$* **As String**)

Parameter *Title$* is the text string to appear across the top of the dialog box.

Example
```
Caption  "TTWIN Login Script"
```
The dialog box will have the title “TTWIN Login Script”.

See also **Begin Dialog**

CDbl ()

Description Converts an expression into a variable of type **Double**.

Declaration **Function CDbl** (**ByVal** *Expression* **As Any**) **As Double**

Parameter *Expression* is any valid numeric or string expression.

Return value Returns *Expression* as type **Double**.

Example
```
CDbl
```

See also **CDbl**, **CLng**, **CSng**, **CStr**, **CVar**

ttCfgNew ()

Description Restores the configuration details to the defaults.

Declaration **Sub CfgNew**

Visual Basic **ttCfgNew** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttCfgNew** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > New

Example `CfgNew`

Opens the default session file `DEFAULT.TWC`, loading the default emulation, comms and file transfer modules.

See also **ttCgfOpen**, **ttCfgSave**, **ttCfgSaveAs**

CgfOpen, **CfgSave**, **CfgSaveAs**

ttCfgOpen ()

Description Sets up configuration details from a specified session file.

Declaration **Sub CfgOpen** (**ByVal** *SessionFile$* **As String**)

Visual Basic **ttCfgOpen** (
ByVal *HSess* **As Long**,
ByVal *SessionFile* **As String**)
As Long

C language **int ttCfgNew**
(**HSess** *HSess*,
LPCSTR *SessionFile*);

Parameter *SessionFile$* is the name of an existing predefined session file, that is, a `.TWC` file.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > Open...

Example
```
CfgOpen  "atlas"
```
Opens the predefined session `ATLAS.TWC`, loading the specified emulation, comms and file transfer modules. Colour, keyboard, display configuration settings, etc., can also be predefined.

Example

See also **ttCgfNew**, **ttCfgSave**, **ttCfgSaveAs**

See also **CgfNew**, **CfgSave**, **CfgSaveAs**

ttCfgSave ()

Description Saves the configuration details in a `.TWC` (predefined session) file.

Declaration **Sub CfgSave**

Visual Basic **ttCfgSave** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttCfgSave** (**HSess** *HSess*);

Remarks The file is saved as either:

- `DEFAULT.TWC`. If you have not loaded an existing predefined session file or used the **SaveAs...** option during the current session.

  OR

- The currently loaded `.TWC` file.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > Save

Example `CfgSave`

Saves the currently open session file.

See also **ttCfgNew**, **ttCgfOpen**, **ttCfgSaveAs**

CfgNew, **CgfOpen**, **CfgSaveAs**

ttCfgSaveAs ()

Description Stores the configuration details in the specified file.

Declaration **Sub CfgSaveAs** (**ByVal** *SessionFile$* **As String**)

Visual Basic

```
ttCfgSaveAs (
ByVal HSess As Long,
ByVal SessionFile As String)
As Long
```

C language **int ttCfgSaveAs** (
HSess *HSess*,
LPCSTR *SessionFile*);

Parameter *SessionFile$* is the name of the predefined session file, into which your current TTWIN configuration details are to be saved.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > SaveAs...

Example
```
CfgSaveAs  "atlas"
```
Saves the current emulation, comms and file transfer module details into `ATLAS.TWC`.

If `ATLAS.TWC` already exists, it is overwritten with the new settings.

Example

See also **CfgNew**, **CgfOpen**, **CfgSave**

See also **ttCfgNew**, **ttCgfOpen**, **ttCfgSave**

ChDir

Description Changes the current default directory.

Declaration **Sub ChDir** (**ByVal** *Path$* **As String**)

Parameter *Path$* specifies the directory that is to be the new default directory. The ChDir does not change the default drive.

If the drive is specified, the default directory for the indicated drive will change but not the default drive.

If a drive is not specified the default directory is changed for the default drive.

Example

```
ChDir d:\documents
```

If the current default drive is `C:` the statement above changes the default directory on the `D:` drive but `C:` remains the default drive.

See also **ChDrive**, **MkDir**, **RmDir**

ChDrive

Description Changes the current drive.

Declaration **Sub ChDrive** (**ByVal** *Path$* **As String**)

Parameter *Path$* specifies the drive that is to be the new default drive.

Example

```
ChDrive c:\
```

Assuming that the current default drive is `D:` the statement above changes the default drive to `C:` .

See also **ChDir**, **MkDir**, **RmDir**

CheckBox

Description Defines a check box.

Syntax **CheckBox** *x, y, cx, cy, Option$, State%*

Remarks A check box allows the user to specify an option. Multiple check boxes can be selected simultaneously.

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of the check box, relative to overall dialog box.

cx, cy are the width (*cx*) and height (*cy*) of the check box region, including *Option$*.

Option$* is the text to be aligned to the right of the check box. An *&* preceding a character will create an ALT shortcut key. That is, for *Option$ ="Te&st", **Te<u>s</u>t** appears beside the check box and ALT_S will toggle the check box state.

State% is an integer variable containing the control state. *State%* is set to one of the following:

State%	Meaning
1	Checked.
0	Not checked.

Example

```
statevar = 1    'check box will appear selected
```

```
CheckBox  40, 53, 30, 20, "&Bold", statevar
```

A check box will appear, see below, in the dialog box with Bold written to the right. The shortcut key, ALT_B, will allow you to select and deselect the Bold check box.

See also **ComboBox**, **GroupBox**, **ListBox**, **OptionButton**, **TextBox**

Chr$ ()

Description Returns a character corresponding to an ASCII value.

Declaration **Function Chr$** (**ByVal** *Asc%* **As Integer**) **As String**
Function Chr (**ByVal** *Asc%* **As Integer**) **As Variant**

Parameter *Asc%* is the ASCII value for the character required.

Return value Returns a character corresponding to *Asc%*.

Example
```
Chr$ (65)
```
Returns the character *A*, the ASCII equivalent.

See also **Asc**, **Str$**

CInt ()

Description Converts an expression to an **Integer**.

Declaration **Function CInt** (**ByVal** *Expression* **As Any**) **As Integer**

Parameter *Expression* is any valid expression. The expression is rounded to the nearest integer value. When the non integer part is equal to 0.5, *Expression* is rounded to an even number.

Return value Returns *Expression* as an **Integer**.

Example
```
NewInt = CInt (95.5)
```
NewInt contains 96.

See also **CDbl**, **CLng**, **CSng**, **CStr**, **CVar**

ttClipboardCopy ()

Description Copies the current selected TTWIN screen region to the clipboard.

Declaration **Sub ClipboardCopy**

Visual Basic **ttClipboardCopy** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttClipboardCopy** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Edit > Copy

Example

```
DispSelectRect 2,3,10,30
ClipboardCopy
```

An area of the screen defined by **DispSelectRect** is copied to the clipboard.

Example

See also **ClipboardPaste**

See also **ttClipboardPaste**

ttClipboardPaste ()

Description Pastes text from the clipboard into the current emulation.

Declaration **Sub ClipboardPaste**

Visual Basic **ttClipboardPaste** (**ByVal** *HSess* **As Long**) **As Long**

int ttClipboardPaste (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Edit > Paste

Example
```
ClipboardPaste
```

Example

See also **ClipboardCopy**

See also **ttClipboardCopy**

CLng ()

Description Converts an expression to a variable of type **Long**.

Declaration **Function CLng** (**ByVal** *Expression* **As Any**) **As Long**

Parameter *Expression* is any valid expression.

Return value Returns *Expression* as a variable of type **Long**.

Example
```
Single = 2,789.9
Long = CLng (single)
```
Long contains 2790.

See also **CDbl**, **CInt**, **CSng**, **CStr**, **CVar**

Close

Description Closes a file(s) opened with an **Open** statement.

Declaration **Sub Close** ([[#]*FileNumber1%* **As String**][,[#]*FileNumber2%* **As String**].....])

Parameter *FileNumber%* is a valid file number. If *FileNumber* is omitted, all files opened with the **Open** statement are closed.

Example

```
Open "FileA.twc" For Input As 1
While Not Eof(1)
  ...
Wend
Close 1
```

See also **FileAttr**, **FreeFile**, **Open**

CloseHostStream

Description Closes the host stream after reading.

Declaration **Sub CloseHostStream**

Remarks This **must** be called after a **ReadHost$** or **ReadHostLine$**.

After an **OpenHostStream** call, all data received from the host is placed into a queue for use by scripts (using **ReadHost**….). Once the script has finished reading, it **MUST** be closed.

Example

```
Dim param$ as String
...
OpenHostStream
      param$ = ReadHostLine$
CloseHostStream
```

See also **OpenHostStream**, **ReadHost$** and **ReadHostLine$**

ttCloseInputCapture ()

Description Closes the host stream.

Visual Basic **ttCloseInputCapture** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttCloseInputCapture** (**HSess** *HSess*);

Remarks This **must** be called after a **ReadInputCapture**.

After an **ttOpenInputCapture** call, all data received from the host is placed into a queue for use by scripts (using **ttReadInputCapture**). Once the script has finished reading, it **MUST** be closed.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.

Value	Meaning
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **OpenInputCapture**, **ReadInputCapture**

ttCloseSession()

Description Closes the handle with the TTWIN session.

Visual Basic **ttCloseSession** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttCloseSessionHandle** (**HSess** *HSess*);

Remarks This subroutine **MUST** be called to close the connection to the active TTWin session. Failure to do so may yield unpredictable results.

The function returns the following values:

Value	Meaning
0	The function succeeded.

Example

See also

ComboBox

Description Defines a combo box.

Declaration **ComboBox** *x, y, cx, cy, Options$, State%*

Remarks A Combo displays only the first of a list of possible options. The remaining options, displayed in the order in which they are specified, are viewed from a drop down menu.

Parameters *x*, *y* are the horizontal (*x*) and vertical (*y*) position of top left corner of the combo box, relative to overall dialog box.

cx, *cy* are the width (*cx*) and height (*cy*) of the combo box region. The control height (*cy*) defines the combined size of the first field as well as the drop down menu.

Options$ is the list of options to be placed in the combo box. This is a single string of tab separated fields. Each field is a line in the combo box.

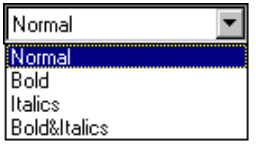
State% is an integer variable identifying the current selection, as follows:

State%	**Option selected**
-1	No selection.
0	First option
1	Second option
etc.	...

Example

```
ComboBox 40, 53, 50, 40, "Normal Bold Italics
Bold&Italics", combovar
```

A combo box will be displayed, see below, containing the four specified entries: **Normal**, **Bold**, **Italics**, **Bold&Italics**.



See also **CheckBox**, **GroupBox**, **ListBox**, **OptionButton**, **TextBox**

ttConnect ()

Description Attempts a connection to the specified host.

Declaration **Sub Connect**

Visual Basic **ttConnect** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttConnect** (**HSess** *HSess*);

Remarks **Connect** attemps a connection to the host specifed in the currently loaded session file.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Action > Connect

Example

```
pause 2
connect
```

Using the current `.TWC` file, a connection is attempted.

Example

See also **Break, Disconnect, IsConnected**

See also **ttBreak**, **ttDisconnect**, ttIsConnected

Cos ()

Description Calculates the cosine of an angle.

Declaration **Function Cos** (**ByVal** *Angle* **As Double**) **As Double**
Function Cos (**ByVal** *Angle* **As Single**) **As Single**

Parameter *Angle* is the angle in radians.

Return value Returns the cosine of *Angle*.

Example
```
x = Cos  '(Radians)
```

See also **Atn**, **Sin**, **Tan**

ttCreateDDEItem ()

Description Creates a DDE item with the given topic and item strings.

Declaration **Sub CreateDDEItem** (**ByVal** *Topic$* **As String**, **ByVal** *Item$* **As String**)

Visual Basic
ttCreateDDEItem
(**ByVal** *HSess* **As Long**,
ByVal *Topic$* **As String**,
ByVal *Item$* **As String**)
As Long

C language **int ttCreateDDEItem**

(**HSess** *HSess*,
LPCSTR *Topic*,
LPCSTR *Item*);

Remarks The topic is automatically created if it does not exist. The initial value is an empty string.

Parameters *Topic$* is the subject of the conversation. (An example topic, for Microsoft Excel would be a `.XLS` file.)

Item$ is the actual item of data being passed. (An example item, for Microsoft Excel would be a cell reference, such as, R1C5.)

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Topic1$ = "System"
Item1$ = "Path"
StartDDEServer
DispClearAll
CreateDDEitem Topic1$, Item1$
```

Create an item *Path* for DDE server *System* topic.

Example

See also **DestroyDDEItem**, **GetDDEItem**, **IsDDEItem**, **SetDDEItem**

See also **ttDestroyDDEItem**, **ttGetDDEItem**, **ttIsDDEItem**, **ttSetDDEItem**

CSng ()

Description Converts an expression to a variable of type **Single**.

Declaration **Function CSng** (**ByVal** *Expression* **As Any**) **As Single**

Parameter *Expression*, is any valid numeric or string expression.

Example

```
x = 96.3490871                          'x is a Double
y = 4.3112963                           'y is a Double
Single = CSng(y) * CSng(x)
```

See also **CDbl**, **CInt**, **CLng**, **CStr**, **CVar**

CStr ()

Description Converts an expression to a string.

Declaration **Function CStr** (**ByVal** *Expression* **As Any**) **As String**

Parameter *Expression* is any valid type. The resulting contents of *Expression* of depend on its initial value.

Expression (initial value)	**Resulting string**
Empty	An empty string, "".
Error	A string containing the word *Error* and the error number
Numeric	The number as a string.

Expression (initial value)	Resulting string
Null	A run-time error occurs.

Return value Returns *Expression* as a string.

Example
```
NumStr = CStr (341.76)
```
NumStr contains “431.76”.

See also **CDbl**, **CInt**, **CLng**, **CSng**, **CStr**, **CVar**

CVar ()

Description Converts an expression to a variable of type **Variant**.

Declaration **Function CVar** (**ByVal** *Expression* **As Any**) **As Variant**

Parameter *Expression*, is any valid numeric or string expression.

Return value Returns *Expression* as a variable of type **Variant**.

See also **CDbl**, **CInt**, **CLng**, **CSng**, **CVar**

DDEAdvise

Description Establishes an advise link with a DDE server.

Declaration **Sub DDEAdvise** (**ByVal** *CNo%* **As Integer**, **ByVal** *Item$* **As String**, *Value$* **As String**[, *Flag%* **As Integer**)

Parameters *CNo* is the conversation handle obtained using the **DDEInitiate** command.

Item$ specifies the item for which the advise link is established. (An example item, for Microsoft Excel would be a cell reference, such as, R1C5.)

When the server sends an advise notice, *Value$* is set to the new value for *Item$*.

Flag% indicates the arrival of an advise notice.

Return value *Flag%*, if provided, is set to 1 when the server sends an advise notice. The intention is that you initially set *Flag%* to 0 and loop waiting for it to change.

Example

```
DDEAdvise  CNo, "R2" + "C4" , NewVar$
```

Waits for the value in cell R2C4 to change, the new value is then passed to *NewVar$*.

See also **DDEExecute**, **DDEInitiate**, **DDEPoke**, **DDERequest$**, **DDETerminate**, **DDETerminateAll, DDEUnAdvise**

DDEExecute

Description Sends a command to be executed by the DDE server.

Declaration **Sub DDEExecute** (**ByVal** *CNo%* **As Integer**, **ByVal** *ExecuteStr$* **As String**)

Parameters *CNo* is the conversation handle obtained using the **DDEInitiate** command.

ExecuteStr$ is an executable operation or command understood by the host application.

Example

```
TRUE  = 1
Shell "c:\excel\excel.exe", 4
CNo = DDEInitiate ("Excel", "Sheet1")
Q$ = Chr$ (34)     ' Set Q$ equal to a quote character i.e. "
DDEExecute CNo, "[OPEN(" + Q$ + "f:\sheet1.xls" + Q$ + ",,TRUE)]"
```

The above statements start Excel and initiates a DDE conversation. Then using **DDEExecute** the worksheet `F:\SHEET1.XLS` is opened.

See also **DDEAdvise**, **DDEInitiate**, **DDEPoke**, **DDERequest$**, **DDETerminate**, **DDETerminateAll, DDEUnAdvise**

DDEInitiate ()

Description Creates a conversation with an application, returning a DDE conversation handle.

Declaration **Function DDEInitiate** (**ByVal** *Service$* **As String**, **ByVal** *Topic$* **As String**) **As Integer**

Parameters *Service$* is the name of the source application or a service name.

Topic$ is the subject of the conversation. (An example topic, for Microsoft Excel would be a `.XLS` file.)

Return value **DDEInitiate** returns the following values:

Value	Meaning
> 0	The conversation handle. (**DDEInitiate** succeeded.)
0	**DDEInitiate** failed.

Example

```
CNo = DDEInitiate ("Excel", "Sheet1.xls")
```

See also **DDEAdvise**, **DDEExecute**, **DDEPoke**, **DDERequest$**, **DDETerminate**, **DDETerminateAll, DDEUnAdvise**

DDEPoke

Description Sends data to a DDE server for the specified item.

Declaration **Sub DDEPoke** (**ByVal** *CNo%* **As Integer**, **ByVal** *Item$* **As String**, **ByVal** *Data$* **As String**)

Parameters *CNo%* is the conversation handle obtained using the **DDEInitiate** command.

Item$ specifies the item to which the data is being sent. (An example item, for Microsoft Excel would be a cell reference, such as, R1C5.)

Data$ is the string sent to DDE server.

Example

```
DDEPoke CNo, "R1" + "C1", "Total"
```

With the *CNo* handle pointing to Microsoft Excel, this example will put the word “Total” into cell R1C1.

See also **DDEAdvise**, **DDEExecute**, **DDEInitiate**, **DDERequest$**, **DDETerminate**, **DDETerminateAll, DDEUnAdvise**

DDERequest$ ()

Description Gets data from a DDE server for the item specified.

Declaration **Function DDERequest$** (**ByVal** *CNo%* **As Integer**, **ByVal** Item*$* **As String**) **As String**

Parameters *CNo%* is the conversation handle obtained using the **DDEInitiate** command.

Item$ specifies the item for which the data is required. (An example item, for Microsoft Excel would be a cell reference, such as, R23C5.)

Return value Retruns the value of the specified item.

Example

```
ExcelPath$ = _
"\\Atlas\programs\MSOffice\Office\EXCEL.EXE"

Shell ExcelPath$, 4
CNo = DDEInitiate("Excel", "Sheet1.xls")
Cell = DDERequest (CNo, "R1C1")
DDETerminate CNo
```

Using TTWIN as the DDE Client, a conversation is established with Microsoft Excel. TTWIN makes a request for contents of the cell R1C1.

See also **DDEAdvise**, **DDEExecute**, **DDEInitiate**, **DDEPoke**, **DDETerminate**, **DDETerminateAll, DDEUnAdvise**

DDETerminate

Description Terminates a DDE conversation.

Declaration **Sub DDETerminate** (**ByVal** *CNo%* **As Integer**)

Parameter *CNo%* is the conversation handle obtained using the **DDEInitiate** command.

Example

```
Command$ = "[OPEN(" + Q$ + "c:\sheet1.xls" + Q$ + ")]"
Shell ExcelPath$, 4
CNo = DDEInitiate("Excel", "Sheet")
...
DDETerminate CNo
```

The above example starts Excel and establishes a conversation. When the the transaction using the specified topic is complete the conversation is terminated.

See also **DDEAdvise**, **DDEExecute**, **DDEInitiate**, **DDEPoke**, **DDERequest$**, **DDETerminateAll, DDEUnAdvise**

DDETerminateAll

Description Terminates all currently available conversation handles.

Declaration **Sub DDETerminateAll**

Example

```
DDETerminateAll
```

See also **DDEAdvise**, **DDEExecute**, **DDEInitiate**, **DDEPoke**, **DDERequest$**, **DDETerminate, DDEUnAdvise**

DDEUnAdvise

Description Terminates an advise link with a DDE server.

Declaration **Sub DDEUnAdvise** (**ByVal** *CNo%* **As Integer**, **ByVal** *Item$* **As String**)

Parameters *CNo* is the conversation handle obtained using the **DDEInitiate** command.

Item$ specifies the item for which the advise link was established.

Example

```
DDEUnAdvise CNo, "R2" + "C4"
```

See also **DDEAdvise**, **DDEExecute**, **DDEInitiate**, **DDEPoke**, **DDERequest$**, **DDETerminate**, **DDETerminateAll**

ttDefineVar ()

Description Defines a variable for DDE.

Visual Basic

ttDefineVar
(**ByVal** *HSess* **As Long**,
ByVal *VarName* **As String**,
ByVal *VType* **As Long)**
As Long

C language

int ttDefineVar
(**HSess** *HSess*,
LPCSTR *VarName*,
int *VType*);

Parameters

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

ttDestroyDDEItem ()

Description Destroys a DDE item with the given topic and item strings.

Declaration **Sub DestroyDDEItem** (**ByVal** *Topic$* **As String**, **ByVal** *Item$* **As String**)

Visual Basic

ttDestroyDDEItem
(**ByVal** *HSess* **As Long**
ByVal *Topic$* **As String**,
ByVal *Item$* **As String**)
As Long

C language

int ttDestroyDDEItem
(**HSess** *HSess*,
LPCSTR *Topic*,
LPCSTR *Item*);

Parameters *Topic$* is the subject of the conversation. (An example topic, for Microsoft Excel would be a `.XLS` file.)

Item$ is the actual item of data being passed. (An example item, for Microsoft Excel would be a cell reference, such as, R1C5.)

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Topic1$ = "System"
```

```
Item$ = "Path"
StartDDEServer
DispClearAll
CreateDDEitem Topic1$, Item$
     ...
     ...
DestroyDDEitem Topic1$, Item$
```

Create an items *Path* for DDE server *System* topic. Before the conversation is terminated the item is destroyed.

Example

See also **CreateDDEItem**, **GetDDEItem**, **IsDDEItem**, **SetDDEItem**

See also **ttCreateDDEItem**, **ttGetDDEItem**, **ttIsDDEItem**, **ttSetDDEItem**

ttDialEntry ()

Description Dials the predefined dial entry.

Declaration **Function DialEntry** (**ByVal** *EntryName$* **As String**) **As Integer**

Visual Basic

ttDialEntry
(**ByVal** *HSess* **As Long**,
ByVal *EntryName* **As String**)
As Long

C language

int ttDialEntry
(**HSess** *HSess*,

LPCSTR *EntryName*);

Parameter *EntryName$* is the dialling entry name.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Result = DialEntry ("DayLight Company Pty")
```

Example

Menu Action > Dial...

See also **DialManual**, **Hangup**, **SelectModem**

See also **ttDial**, **ttDialManual**, **ttHangup**, **ttSelectModem**

ttDialManual ()

Description Dials a specified modem number using the current configured modem.

Declaration **Function DialManual** (**ByVal** *Number$* **As String** [, **ByVal** *Description$* **As String**]) **As Integer**

Visual Basic

ttDialManual
(**ByVal** *HSess* **As Long**,
ByVal *Number* **As String**,

ByVal *Description* **As String**)
As Long

C language

int ttDialManual
(**HSess** *HSess*,
LPCSTR *Number*,
LPCSTR *Description*);

Parameters

Number$ is the modem number to be dialled using the current configured modem.

Description$ is, if provided, is displayed in the progress tile.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu

Action > Dial...

Example

```
Result = DialManual ("022813005", "Life Company Pty")
```

Example

See also

DialEntry, **Hangup**, **SelectModem**

See also

ttDial, **ttDialEntry**, **ttHangup**, **ttSelectModem**

Dialog

Description Displays a dialog box for data entry.

Syntax **Dialog** *DlgBox*

Parameter *DlgBox* must be a variable declared as a dialog box type. The dialog box type is defined using the **Begin Dialog...End Dialog** statements.

The data entered into the dialog box is stored in the variables defined within the dialog box type definition.

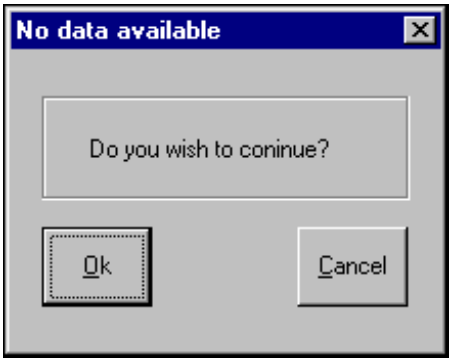
If the cancel button is pressed, a run-time error is produced that can be trapped with **On Error**.

Example

```
Option Explicit
Dim Response As Integer
Dim Prompt As String

Prompt = "Do you wish to continue?"

Begin Dialog ContinueBox 220, 140, 140, 90
     Caption "No data available"
     GroupBox 10, 10, 120, 35,""
     Text 25, 25, 100, 10, Prompt
     OKButton 10, 53, 36, 24
     CancelButton 93, 53, 36, 24
     ButtonGroup Response
End Dialog

Dim ContinuePrompt As ContinueBox
```

```
Dialog (ContinuePrompt)
If Response = -1 Then       'Cancel button was pressed
...
ElseIf Response = 0 Then        'OK button was pressed
...
End If
```

In the above example, first the dialog box type *ContinueBox* is defined, then the variable *ContinuePrompt* is declared. After which the dialog box *ContinuePrompt* is displayed, see below. Finally, the return value from the **Dailog** function is tested to determine the action to be taken.



See also **Begin Dialog...End Dialog**, **Button**, **ButtonGroup**, **CancelButton CheckBox**, **ComboBox**, **GroupBox**, **Dialog**, **ListBox**, **OKButton**, **On Error**, **OptionButton**, **OptionGroup**, **Text**, **TextBox**

Dim

Description Declares variables and arrays, and allocates the required memory space.

Syntax **Dim** *Variable* [([*LowerBound* **To**] *UpperBound*)] [**As** *Type*]

Parameters *Variable* is the name of the variable being declared.

The *LowerBound*, if used is the number of the first element in the array. That is, if the *LowerBound* is 3, then the first array element is 3 and there are no elements 1 or 2.

When *LowerBound* is not specified, a default of 0 is used.

The *UpperBound* is the number of elements in the array only when *LowerBound* is 1.

Arrays may be multi-dimensional, the total number of array elements is limited only by memory. All elements must live in one 64K memory block.

Type is one of the supported data types. When *Type* is not specified, **Integer** is assumed.

Example 1

```
DIM IntVar As Integer
```

Declares the **Integer** variable, *IntVar*.

Example 2

```
DIM StringVar As String
```

Declares a **String** variable called *StringVar*.

Example 3

```
DIM LargeSize (8 To 13) As String
```

This declares an array of 6 elements called *LargeSize* of type **String**. The first element of the array is 8.

Example 4 `DIM SmallArray (4)`

This declares an array, *SmallArray*, of 5 integer elements. The first element being 0.

Note: *If no data type is specified,* ***Integer*** *is assumed.*

Example 5 `DIM ChessBoard (1 TO 8, 1 TO 8)`

This declares a two dimensional integer array called *ChessBoard*. The array contains 64 elements, i.e., (1,1), (1,2) ... (8,7), (8,8). The first element is (1,1). The 14th element is (2,6).

See also **ReDim**

ttDisconnect ()

Description Disconnect from the remote host.

Declaration **Sub Disconnect**

Visual Basic **ttDisconnect** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttDisconnect** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Action > Disconnect

Example `If IsConnected Then Disconnect`

Example

See also **Break, Connect, IsConnected**

See also **ttBreak, ttConnect, ttIsConnected**

ttDispClearAll ()

Description Clears the emulation screen.

Declaration **Sub DispClearAll**

Visual Basic **ttDispClearAll** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttDispClearAll** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Edit > Clear Display

Example

```
DisplClearAll
```

Example

See also **DispSelectAll**, **DispClearSelect**

See also **ttDispSelectAll**, **ttDispClearSelect**

ttDispClearSelect ()

Description Clears the selected screen region.

Declaration **Sub DispClearSelect**

Visual Basic **ttDispClearSelect** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttDispClearSelect** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Edit > Clear All

Example `DisplClearAll`

Example

See also **DispClearAll**, **DispSelectAll**

See also **ttDispClearAll**, **ttDispSelectAll**

DispHeight ()

Description Gets the current emulation screen height in rows.

Declaration **Function DispHeight As Integer**

Return value Returns the screen height for current emulation in rows.

Example

```
CurrHeight = DispHeight
```

If you are currently running a 24 x 80 display then *CurrHeight* will be contain 24.

See also **DispWidth**

Display

Description Sends data to the emulation screen as if it came from the host.

Declaration **Sub Display** (**ByVal** *Str$* **As String**)

Parameter *Str$* is any valid string including emulation specific escape sequences.

Example

```
BoxOff = Chr$ (27) + "(B"
...
Display BoxOff
```

In the above example, an escape sequence for a DEC terminal is sent to the screen as if from the host.

See also **SendKeys**

ttDispReadAttr ()

Description Reads the attribute values, from the screen, for a string of a given length.

Declaration **Sub DispReadAttr** (*Attr%()* **As Integer**, **ByVal** *Row%* **As Integer**, **ByVal** *Col%* **As Integer**, **ByVal** *Length%* **As Integer**)

Visual Basic

ttDispReadAttr
(**ByVal** *HSess* **As Long**,
Attr **As Long**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**,
ByVal *Length* **As Long**)
As Long

C language

int ttDispReadAttr
(**HSess** *HSess*,
DWORD FAR **Attr*,
int *Row*,
int *Col*,
int *Length*);

Parameters

Attr%() is an integer array of the returned attribute values. The possible attribute values are combinations of the following:

Attr%	**Attribute**
0	Normal
1	Blink
2	Underline
4	Bold
8	Dim
16	Reverse
32	Overline
64	Strike out
128	Invisible

Row% the start line of the attribute read.

Col% the start column of the attribute read.

Note: *Screen coordinates have an origin of (1,1) at the top left of the screen.*

Length% the length of the string to be read, in characters.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
DIM AttrArray (500)
DispReadAttr  AttrArray%(), 10, 1, 160
```

Read 160 characters starting from the first character on line 10. If the screen is an 80 column display, lines 10 and 11 will be read.

And, if the 5th character on line 11has the attributes **Bold**, **Underline** and **Overline**, then *AttArray(85)* will be 38, i.e., 4 (Bold) + 2 (Underline) + 32 (Overline).

Example

See also **DispReadText**

See also **ttDispReadText**

ttDispReadText ()

Description Reads a given length character string from the screen.

Declaration **Sub DispReadText** (*Text$* **As String**, **ByVal** *Row%* **As Integer**, **ByVal** *Col%* **As Integer**, **ByVal** *Length%* **As Integer**)

Visual Basic

ttDispReadText
(**ByVal** *HSess* **As Long**,
ByVal *Text* **As String**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**,
ByVal *Length* **As Long**)
As Long

C language

int ttDispReadText
(**HSess** *HSess*,
LPSTR *Text*,
int *Row*,
int *Col*,
int *Length*);

Parameters

Text$ is a string array holding the read character string.

Row% is the starting line of the character string read.

Col% is the starting column of the character string read.

Note: *Screen coordinates have an origin of (1,1) at the top left of the screen.*

Length% is the length of the sting to be read, in characters.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
DIM StrArray$ (500) AS String
DispReadText  StrArray$(), 10, 1, 160
```

Reads 160 characters starting from the first character on line 10 and put them in *StrArray$ ()*.

If the screen is 80 column display then lines 10 and 11 will be read.

Example

See also **DispReadAttr**

See also **ttDispReadAttr**

ttDispSelectAll ()

Description Selects the entire screen for copy or paste operations.

Declaration **Sub DispSelectAll**

Visual Basic **ttDispSelectAll** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttDispSelectAll** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Edit > Clear Display

Example `DispSelectAll`

Example

See also **DispClearAll**, **DispClearSelect**

See also **ttDispClearAll**, **ttDispClearSelect**

ttDispSelectCont ()

Description Selects a continuous region from a specified starting point through to a given ending point for copy and paste operations.

Declaration **Sub DispSelectCont** (**ByVal** *Row1%* **As Integer**, **ByVal** *Col1%* **As Integer**, **ByVal** *Row2%* **As Integer**, **ByVal** *Col2%* **As Integer**)

Visual Basic

ttDispSelectCont
(**ByVal** *HSess* **As Long**,
ByVal *Row1* **As Long**,
ByVal *Col1* **As Long**,
ByVal *Row2* **As Long**,
ByVal *Col2* **As Long**)
As Long

C language

int ttDispSelectCont
(**HSess** *HSess*,
int *Row1*,
int *Col1*,
int *Row2*,
int *Col2*);

Parameters

Row1% is the starting row (line) of the continuous region select.

Col1% is the starting column of the continuous region select.

Row2% is the ending row of the continuous region select.

Col2% is the ending column of the continuous region select.

Note: *Screen coordinates have an origin of (1,1) at the top left of the screen.*

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
DispSelectCont 10,40,15,20
```

A continuous region from row 10, column 40 to row 15, column 20 is selected (highlight).

Example

See also **DispSelectRect**

See also **ttDispSelectRect**

ttDispSelectRect ()

Description Selects the rectangular region from specified starting point to a given ending point for copy and paste operations.

Declaration **Sub DispSelectRect** (**ByVal** *Row1%* **As Integer**, **ByVal** *Col1%* **As Integer**, **ByVal** *Row2%* **As Integer**, **ByVal** *Col2%* **As Integer**)

Visual Basic

ttDispSelectRect
(**ByVal** *HSess* **As Long**,
ByVal *Row1* **As Long**,
ByVal *Col1* **As Long**,
ByVal *Row2* **As Long**,
ByVal *Col2* **As Long**)
As Long

C language

int ttDispSelectRect (
HSess *HSess*,
int *Row1*,
int *Col1*,
int *Row2*,
int *Col2*);

Parameters *Row1%* is the starting row (line) of the rectangular region select.

Col1% is the starting column of the rectangular region select.

Row2% is the ending row of the rectangular region select.

Col2% is the ending column of the rectangular region select.

Note: *Screen coordinates have an origin of (1,1) at the top left of the screen.*

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
DispSelectRect 10,30,15,55
```

A rectangular region from row 10, column 30 to row 15, column 55 is selected.

Example

See also **DispSelectCont**

See also **ttDispSelectCont**

DispWidth ()

Description Get the current emulation screen width in columns.

Declaration **Function DispWidth As Integer**

Return value Returns the screen width for the current emulation in columns.

Example

```
CurrWidth = DispWidth
```

If you are currently running a 24 x 80 display then *CurrWidth* will be contain 80.

See also **DispHeight**

Do...Loop

Description Repeats a group of statements based on a condition.

Syntax 1

Do [**While** | **Until** *Condition*]
[*Statements*]
Loop

Syntax 2

Do
[*Statements*]
Loop [**While** | **Until** *Condition*]

Remarks There are 2 forms of the **Do...Loop**. The loop can be either executed while a condition is true or until a condition becomes true.

Parameter *Condition* is any numeric or string expression. The *Statements* in the **Do...Loop** are repeatedly executed either **While** *Condition* is true or **Until** *Condition* is becomes true.

You can use an **Exit Do** statement as an alternate way to exit the **Do...Loop**. Any number of **Exit Do** statements can be placed anywhere in *Statements.*

Example

```
Count = 10
Do
      ...
```

```
        Count = Count - 1
Loop Until Count = 1
```

In the above example, the **Do...Loop** is executed nine times.

ttEmKeys ()

Description Sends a sequence of keystrokes to the host as if typed at a keyboard.

Declaration **Sub EmKeys** (**ByVal** *Text$* **As String**)

Visual Basic **ttEmKeys** (**ByVal** *HSess* **As Long**,
ByVal *Text* **As String**,
ByVal *Length* **As Long**)
As Long

C language **int ttEmKeys** (
HSess *HSess*,
LPCSTR *Text*,
unsigned *Length*);

Parameters *Text %* is the sequence of keystrokes sent to the host.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example The following sends a command to a DEC VT220 emulation with a host running UNIX:

```
keys$ = " ls -l {Return}"
EmKeys (keys$)
```

See also **PhysicalKey**, **SendKeys**, **VirtualKey**

See also **ttKeyText**, **ttPhysicalKey**, **ttVirtualKey**

ttEmReset ()

Description Resets the emulation.

Declaration **Sub EmReset**

Visual Basic **ttEmReset** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttEmReset** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Action > Reset terminal

Example

```
EmRest
```

Example

See also

Environ$ ()

Description Returns the value of an environment variable.

Declaration **Function Environ$** (**ByVal** *Var$* **As String**) **As String**

Parameter *Var$* is a declared environment variable. It is not case sensitive. If *Var$* in not in the environment an empty string is returned.

Return value Returns the value of *Var$*.

Example

```
StrVar$ = Environ$ ("PATH")
```

Returns the TTWIN PC's current DOS PATH setting.

EOF()

Description Indicates whether the end of a file has been reached.

Declaration **Function EOF** (**ByVal** *FileNumber%* **As Integer**) **As Integer**

Parameter *FileNumber%* is any valid file number. The file must be opened using the **Open** statement.

Return value The possible return values are:

Value	Meaning
1	(True) End of file has been reached
0	(False) Not end of file

See also **LOF**

Erl ()

Description Returns the line number of an error.

Declaration **Function Erl As Integer**

Return value Returns the line number of the error.

Example

```
Line = Erl ()
```

See also **Assert**, **Err**, **Error$**

Err ()

Description Returns an error number.

Declaration **Function Err As Integer**

Return value Returns the error number.

Example `ErrorNo = Err ()`

See also **Assert**, **Erl**, **Error$**

Error$ ()

Description Returns the description of the an error.

Declaration **Function Error$ As String**

Return value Returns the description of the most recent error.

Example

```
On Error GoTo ErrRoutine
...
ErrRoutine:
     Line$ = Str$(Erl)
     ErrNo$ = Str$(Err)
     MsgBox "Line number = " + Line$ + Chr$(13) + _
     "Error number = " + ErrNo$ + Chr$(13) +
     "Description:" + Error$, 16, "Error"
     ...
     Resume Next
```

See also **Assert**, **Erl**, **Err**

ttExit ()

Description Closes the current TTWIN session.

Visual Basic **ttExit** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttExit** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > Exit

Example

Exp ()

Description Calculates e (natural logarithm) raised to a power of the given number.

Declaration **Function Exp** (**ByVal** *Number* **As Double**) **As Double**
Function Exp (**ByVal** *Number* **As Single**) **As Single**

Parameter *Number* is any valid numeric expression.

Return value Returns e (natural logarithm) raised to a power of *Number*.

Example `ExpValue = Exp (100)`

FileAttr ()

Description Returns the operating system handle or file mode for a file opened with an **Open** statement.

Declaration **Function FileAttr** (**ByVal** *FileNumber%* **As Integer**, **ByVal** *FileType%* **As Integer**) **As Integer**

Parameters *FileNumber%* is any valid file number.

FileType% indicates the type of information required.

FileType%	Information required
1	File mode
2	Operating system handle

Return value When *FileType%* = 1, the possible return values are:

Value	File type
1	Input
2	Output
4	Random
8	Append
32	Binary

Example

```
Open "FileA.twc" For Input As 1
     ...
Mode = FileAttr (1, 1)
```

```
If Mode = 1 Then
     ...
ElseIf Mode = 8 Then
     ...
Else
     ..
End  If
```

See also **Close**, **FreeFile**, **Open**

FileGet ()

Description Gets a file from the remote host.

Declaration **Function FileGet** (**ByVal** *RemoteName$* **As String**[, **ByVal** *LocalName$* **As String**]) **As Integer**

Remarks The file is transferred using the currently loaded TTWIN file transfer module. The available file transfer protocols are: ASCII, External, FTP Drag and Drop, IBM IND$FILE, Kermit, TTWIN FTP, XMODEM, YMODEM, and ZMODEM.

Note: *With some file transfer modules the protocol must also be running on the server on the remote host when this command is issued.*

Parameters *RemoteName$* is the name of the file being retrieved from the remote host system.

LocalName$ (optional) is the name the file will be given when saved on the local system.

Return value The value returned by the function is as follows:

Value	Meaning
< 0	Transfer failed.
>= 0	Transfer succeeded.

Example

```
FileName$ = "TEST.EXE"
SendData "kermit -x " + Chr$(13)
If WaitForString (10, "Kermit server starting.") = 1
Then
     Pause 1
     If FileGet (FileName$) >= 0 Then
          'Transfer successful
     End If
End If
```

See also **FileReceive**, **FileSend**

FileReceive ()

Description Receives a file from the remote host.

Declaration **Function FileReceive** ([**ByVal** *LocalName$* **As String**]) **As Integer**

Remarks The file is transferred using the currently loaded TTWIN file transfer module. The available file transfer protocols are: ASCII, External, FTP Drag and Drop, IBM IND$FILE, Kermit, TTWIN FTP, XMODEM, YMODEM, and ZMODEM.

Note: *With some file transfer modules the protocol must also be running on the server on the remote host when this command is issued.*

Parameter *LocalName$* (optional) is the name the file on the local system being sent. The local filename is only needed if the filename is not specified when the remote file transfer server is started.

Return value The value returned by the function is as follows:

Value	Meaning
< 0	Transfer failed.
>= 0	Transfer succeeded.

Example

```
FileName$ = "TEST.EXE"
SendData "kermit -s " + FileName$ + Chr$(13)
If WaitForString (10, "give a RECEIVE command...") = 1
Then
          If FileReceive (FileName$) >= 0 Then
          ' Transfer successful
     End If
End If
```

See also **FileGet**, **FileSend**

FileSend ()

Description Sends a file to the remote host.

Declaration **Function FileSend** (**ByVal** *LocalName$* **As String** [, **ByVal** *RemoteName$* **As String**])**As Integer**

Remarks The file is transferred using the currently loaded TTWIN file transfer module. The available file transfer protocols are: ASCII, External, FTP Drag and Drop, IBM IND$FILE, Kermit, TTWIN FTP, XMODEM, YMODEM, and ZMODEM.

Note: *With some file transfer modules the protocol must also be running on the server on the remote host when this command is issued.*

Parameters *LocalName$* is the name the file will be given when saved onto the local system.

RemoteName$ (optional) is the name of the file being retrieved from the remote host system.

Return value The value returned by the function is as follows:

Value	Meaning
< 0	Transfer failed.
>= 0	Transfer succeeded.

Example

```
FileName$ = "TEST.EXE"
SendData "kermit -r " + FileName$ + Chr$(13)
If WaitForString (10, "give a SEND command...") = 1 Then
     If FileSend (FileName$) >= 0 Then
          Transfer successful
     End If
End If
```

See also **FileGet**, **FileReceive**

Fix ()

Description Returns the integer part of a number.

Declaration

Function (**ByVal** *Num* **As Double**) **As Double**
Function (**ByVal** *Num* **As Long**) **As Long**
Function (**ByVal** *Num* **As Integer**) **As Integer**
Function (**ByVal** *Num* **As Single**) **As Single**
Function (**ByVal** *Num* **As Variant**) **As Variant**

Parameter *Num* is any valid numeric expression.

Return value **Fix** removes the fractional portion of a number, returning an integer, as follows:

Num	Returned integer
< 0	>= *Num*
>= 0	<= *Num*

Example 1

```
IntNumber = Fix (91.2)
```

IntNumber contains 91.

Example 2

```
IntNumber = Fix (-79.12)
```

IntNumber contains -79.

Example 3

```
IntNumber = Fix (-56.73)
```

IntNumber contains -56.

See also **Int**

For...Next

Description Executes a group of statements a designated number of times.

Syntax **For** *CountVar = StartExp* **To** *EndExp* [**Step** *StepExp*]
Statement(s)
Next [*CountVar*]

Parameters The first time the loop is executed the counter variable, *CountVar*, is set to the value of *StartExp.* The value of *CountVar* is then compared with that of *EndExp.*

If *CountVar* <= *EndExp* then the *Statement(s)* are executed.

If *CountVar* > *EndExp* then the *Statement(s)* are not executed.

StepExp, is the amount the count variable increments on each loop. If *StepExp is* omitted, the counter is incremented by 1.

On reaching **Next**, control goes back to the start of the **For** statement, where the value of *StepExp*, is added to *CountVar*. The same comparison of *CountVar* and *EndExp* is then done and the loop is conditionally executed.

The counter variable following the **Next** keyword is optional and if not present the counter variable of the last encountered **For** expression is used.

Example 1

```
For i = 1 To 5
     Beep
Next i
```

This will call the **Beep** routine 5 times. On exit *i* = 6.

Example 2

```
For i = 1 To 10 Step 4
     Display (Str$(i))
Next i
```

This will display the values 1, 5 and 9. On exit *i* = 13.

Example 3

```
For i = 100 To 1 Step -1
     Display (Str$ (i))
Next
```

This will display the values 100, 99...1. On exit *i* = 0.

FreeFile ()

Description Returns the next file number.

Declaration **Function FreeFile As Integer**

Return value **FreeFile** returns the next unused file number in a range from 1 to 511 inclusive.

Example

```
FileNo = FreeFile
Open "FileX.twc" For Input As FileNo
```

This example finds the next available file number and assigns the number to the next file opened.

See also **Close**, **FileAttr**, **Open**

ttFtGet ()

Description Gets a file from the remote host.

Visual Basic

ttFtGet
(**ByVal** *HSess* **As Long**,
ByVal *RemoteName* **As String**,
ByVal *LocalName* **As String**)
As Long

C language

int ttFtGet
(**HSess** *HSess*,
LPCSTR *RemoteName*,
LPCSTR *LocalName*);

Remarks The file is transferred using the currently loaded TTWIN file transfer module. The available file transfer protocols are: ASCII, External, FTP Drag and Drop, IBM IND$FILE, Kermit, TTWIN FTP, XMODEM, YMODEM, and ZMODEM.

Note: *With some file transfer modules the protocol must also be running on the server on the remote host when this command is issued.*

Parameters *RemoteName* is the name of the file being retrieved from the remote host system.

LocalName (optional) is the name the file will be given when saved on the local system.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttFtReceive**, **ttFtSend**

ttFtReceive ()

Description Receives a file from the remote host.

Visual Basic

ttFtReceive
(**ByVal** *HSess* **As Long**,
ByVal *LocalName* **As String**)
As Long

C language

int ttFtReceive
(**HSess** *HSess*,
LPCSTR *LocalName*);

Remarks The file is transferred using the currently loaded TTWIN file transfer module. The available file transfer protocols are: ASCII, External, FTP Drag and Drop, IBM IND$FILE, Kermit, TTWIN FTP, XMODEM, YMODEM, and ZMODEM.

Note: *With some file transfer modules the protocol must also be running on the server on the remote host when this command is issued.*

Parameter *LocalName* is the name the file will be saved as on the local system. The local filename is only needed if the filename is not specified when the remote file transfer server is started.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttFtGet**, **ttFtSend**

ttFtSend ()

Description Sends a file to the remote host.

Visual Basic

ttFtSend
(**ByVal** *HSess* **As Long**,
ByVal *LocalName* **As String**,
ByVal *RemoteName* **As String**)
As Long

C language

int ttFtSend
(**HSess** *HSess*,
LPCSTR *LocalName*,
LPCSTR *RemoteName*);

Remarks The file is transferred using the currently loaded TTWIN file transfer module. The available file transfer protocols are: ASCII, External, FTP Drag and Drop, IBM IND$FILE, Kermit, TTWIN FTP, XMODEM, YMODEM, and ZMODEM.

Note: *With some file transfer modules the protocol must also be running on the server on the remote host when this command is issued.*

Parameters *LocalName* is the name of the file that will be transferred.

RemoteName is the name the file will be saved as on the remote host system.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttFtGet**, **ttFtReceive**

GetArgs ()

Description Gets the arguments for a script.

Declaration **Function GetArgs As String**

Remarks This function is used to pass a value to a variable within a script. The string passed to the script can be parsed using any of the string manipluation functions.

Example If the string

`\S"Login.tsl john"`

is entered in the **Startup string** field in Session Preferences and **Auto connect on statrtup** is selected then the user name "john" is passed to the script `LOGIN.TSL` as follows:

```
User$ = GetArgs              '"john" is assigned to User$
Password$ = Lcase$ ("password")
Failed = 1
pause 2
```

```
        if waitforstring (1,"login:") = 1 then
            SendData User$ + Chr$(13)
            if waitforstring (10, "Password:") =1 then
                SendData Password$ + Chr$(13)
                Failed = 0
            end if
        end if
    if Failed = 1 then TTWinExit
```

See also

ttGetCharDetail ()

Description

Visual Basic **ttGetCharDetail**
(**ByVal** *HSess* **As Long**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**,
ByVal *FgColour* **As Long**,
ByVal ***B****gColour* **As Long**,
ByVal *Features* **As Long**)
As Long

C language

int ttGetCharDetail
(**HSess** *HSess*,
int *Row*,
int *Col*,
unsigned FAR **FgColour*,
unsigned FAR **BgColour*,
unsigned FAR **Features*);

Parameter

Row,
Col,
FgColour,
BgColour,
Features);

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

GetCursorCol ()

Description

Gets the column for the current cursor position.

Declaration

Function GetCursorCol As Integer

Return value Returns the column for the current cursor position. The leftmost column returns a value of 1.

Example

```
CurrCol = GetCursorCol
```

The current cursor column position is assigned to the integer variable *CurrCol*.

See also **GetCursorRow**

ttGetCursorPosn ()

Description Gets the row and column values for the current cursor position.

Visual Basic

ttGetCursorPosn
(**ByVal** *HSess* **As Long**,
Row **As Long**,
Col **As Long**)
As Long

C language

int ttGetCursorPosn
(**HSess** *HSess*,
int FAR **Row*,
int FAR **Col*);

Parameters The row and column values are assigned to *Row* and *Col* respectively. The top row and leftmost column both return a value of 1.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

GetCursorRow ()

Description Gets the row of the current cursor position.

Declaration **Function GetCursorRow As Integer**

Return value Returns the row for the current cursor position. The top row returns a value of 1.

Example

```
CurrRow = GetCursorRow
```

The current cursor row position is assigned to the integer variable *CurrRow*.

See also **GetCursorCol**

ttGetCursorStyle ()

Description Returns th e cursor style.

Declaration **Function GetCursorStyle As Integer**

Visual Basic
ttGetCursorStyle
(**ByVal** *HSess* **As Long**,
Style **As Long**)
As Long

C language
int ttGetCursorStyle
(**HSess** *HSess*,
unsigned FAR **Style*);

Parameter *Style* is

Return value The possible returns values are:

Value	Cursor style
&h00	Leave the cursor unchanged. (The default is cursor on and blinking.)
&h01	Full block cursor.
&h02	Small block cursor.
&h03	Horizontal line cursor.
&h04	Vertical line cursor.
&h40	Not visible, (visible otherwise).
&h80	Steady cursor, (blinking otherwise).

These return values are declared as constants in the header files; `TWCONST.H` and `TWDECL.BAS`. Refer to *Appendix B -*

Constants on page 381 for a complete list of declared constants.

Example `CurrStyle = GetCursorStyle`
The current cursor style is returned via the integer variable *CurrStyle*.

Example

See also **SetCursorStyle**

See also **ttSetCursorStyle**

ttGetDDEItem$ ()

Description Returns the current value of the specified DDE item when the DDE client performs a poke.

Declaration **Function GetDDEItem$** (**ByVal** *Topic$* **As String**, **ByVal** *Item$* **As String**) **As String**

Visual Basic
ttGetDDEItem
(**ByVal** *HSess* **As Long**
ByVal *Topic$* **As String**,
ByVal *Item$* **As String**)
As Long

C language
int ttGetDDEItem
(**HSess** *HSess*,
LPCSTR *Topic*,
LPCSTR *Item*,

LPSTR *Buff*,
unsigned *BuffLength*);

Parameters

Topic$ is the subject of the conversation. (An example topic, for Microsoft Excel would be a `.XLS` file.)

Item$ is the actual item of data being passed. (An example item, for Microsoft Excel would be a cell reference, such as, R1C5.)

Return value

Returns the value of the DDE item specified by *Topic$* and *Item$*.

Return value The function returns the following values:

Value	Meaning
>0	The specified DDE item.
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
GetDDEitem Topic1$, Item1$
```

Example

See also **CreateDDEItem**, **DestroyDDEItem**, **IsDDEItem**, **SetDDEItem**

See also **ttCreateDDEItem**, **ttDestroyDDEItem**, **ttIsDDEItem**, **ttSetDDEItem**

ttGetFieldAttr ()

Description Returns the attributes of a field.

Visual Basic

ttGetFieldAttr
(**ByVal** *HSess* **As Long**,
ByVal *StartRow* **As Long**,
ByVal *StartCol* **As Long**,
FieldAttr **As Long**)
As Long

C language

int ttGetFieldAttr (
HSess *HSess*,
int *StartRow*,
int *StartCol*,
unsigned FAR **FieldAttr*);

Parameter

StartRow and *StartCol* identify the field which the attributes are returned in the *FieldAttr* parameter.

Note: *StartRow/Start Col* do not need tok be at the beginning of the field, but may be anywhere in the field area.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

ttGetFieldData

ttGetFieldData ()

Description Returns the contents of a field. Only available for a limited set of emulations.

Visual Basic

ttGetFieldData
(**ByVal** *HSess* **As Long**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**,
ByVal *Buff* **As String,**
ByVal *BuffLength* **As Long**)
As Long

C language

int ttGetFieldData
(**HSess** *HSess*,
int *Row*,
int *Col*,
LPSTR *Buff*,
unsigned *BuffLength*);

Parameter *Row* and *Col* are passed to the function.

Buff points to an area where the field addressed by *Row/Col* is stored.

Note: *Row/Col* do not need to be at the beginning of the field, but may anywhere in the field area.

BuffLength is the length of the area *Buff* points to.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.

Value	Meaning
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttGetFieldAttr**

ttGetIniFile$ ()

Description Gets the initialisation filename, i.e., the `.INI` file.

Declaration
Function GetIniFile$ As String
Function GetIniFile As Variant

Visual Basic
ttGetIniFile
(**ByVal** *HSess* **As Long 32 Bit**,
ByVal *Buff* **As String,**
ByVal *BuffLength* **As Long**)
As Long

C language
int ttGetIniFile
(**HSess** *HSess*,
LPSTR *Buff*,
unsigned *BuffLength*);

Parameters *Buffer* for the full path of the initialisation file.

BuffLength is the length of the array.

Return value Returns the full path of the initialisation file.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Path = GetIniFile$
```

Example

See also **GetSessionFile$**

See also **ttGetSessionFile$**

ttGetScreenSize ()

Description Gets the number of rows and columns for the current emulation screen.

Visual Basic

ttGetScreenSize
(**ByVal** *HSess* , **As Long**
ScrHeight **As Long,**
ScrWidth **As Long**)
As Long

C language

int ttGetScreenSize
(**HSess** *HSess*,

int FAR **ScrHeight*,
int FAR **ScrWidth*);

Parameters *ScrWidth* is number of columns.

ScrHeight is number of rows.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

ttGetScriptPath ()

Description Gets the full path for the location (directory) for the script files.

Visual Basic
ttGetScriptPath
(**ByVal** *HSess* **As Long**,
ByVal *Buff* **As String**,
BuffLength **As Long**)
As Long

C language
int ttGetScriptPath
(**HSess** *HSess*,

LPSTR *Buff*,
unsigned *BuffLength*);

Parameters

Buff for the full path of the scripts directory.

BuffLength is the size of the array *Buff*.

Return value

The function returns the following values:

Value	Meaning
0 -	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

ttOpenSessionByID ()

Description Opens the TTWin session by the ID session string.

Visual Basic **ttOpenSessionByID** (**ByVal** *IDString* **As String**) **As Long**

C language **HSESS ttOpenSessionByID** (**LPCSTR** *IDString*);

Remarks The session ID is set using the **/l** switch when starting TTWIN.

Parameter *IDString* is session ID.

Return value The function returns HSESS to session:

Value	Meaning
0	The function failed.

Example

See also **ttOpenSessionByTitle**, **ttGetSessionCount**, **ttGetSessionFile$**, **ttGetSessionList**

ttOpenSessionByTitle ()

Description Opens the session by the session title.

Visual Basic **ttOpenSessionByTitle** (**ByVal** *Title* **As String**) **As Long**

C language **HSESS ttOpenSessionByTitle** (**LPCSTR** *Title*);

Parameter *Title* is session title set using **ttSetTitle**

Return value The function returns HSESS to the session:

Value	Meaning
0	The function failed.

Example

See also **ttGetSessionByID**, **ttGetSessionCount**, **ttGetSessionFile$**, **ttGetSessionList**

ttGetSessionCount ()

Description Returns the number of active sessions.

Declaration **Function GetSessionCount As Integer**

Visual Basic **ttGetSessionCount** () **As Long**

C language **int ttGetSessionCount** (**void**);

Return value Returns the number of active sessions.

Return value The function returns the following values:

Value	Meaning
> = 0	The number of active sessions.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
count = GetSessionCount
If count <> 0 Then
     ...
```

Example

See also **GetSessionFile$**, **GetSessionList**

See also **ttGetSessionByID**, **ttGetSessionByTitle**, **ttGetSessionFile$**, **ttGetSessionList**

ttGetSessionFile$ ()

Description Gets the session filename, i.e., the `.TWC` file.

Declaration
Function GetSessionFile$ As String
Function GetSessionFile As Variant

Visual Basic
ttGetSessionFile
(**ByVal** *HSess* **As Long**,
ByVal *Path* **As String**,
ByVal *BuffSize* **As Long**)
As Long

C language
int ttGetSessionFile
(**HSess** *HSess*,
LPSTR *Path*,
unsigned *BuffSize*);

Parameters *Path* buffer for the full path of the session file.

BuffSize is the size of *Path*.

Return value Returns the full path of the active session file.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Path$ = GetSessionFile$
```

See also **GetSessionCount**, **GetSessionList**

See also **ttGetIntFile**, **ttGetSessionByID**, **ttGetSessionByTitle**, **ttGetSessionCount**.

ttGetStatusMsg$ ()

Description Gets the status message.

Declaration
Function GetStatusMsg$As String
Function GetStatusMsg As Variant

Visual Basic
ttGetStatusMsg
(**ByVal** *HSess* **As Long**,
ByVal *Buff* **As String**,
ByVal *BuffSize* **As Long**)
As Long

C language
int ttGetStatusMsg
(**HSess** *HSess*,
LPSTR *Buff*,
unsigned *BuffSize*).

Remarks This message is displayed on the Bottom status bar when no system state messages are displayed (e.g., dialling).

Parameters *Buff* is the buffer for the status message.

BuffSize is the length of the array.

Return value Returns the message displayed on the bottom status line.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
BottomMsg = GetStatusMsg
```

The Bottom status line contents are returned via the string variable *BottomMsg*.

Example

```
dll
```

See also **SetStatusMsg**

See also **ttSetStatusMsg**

ttGetTitle$ ()

Description Gets the title of the current TTWIN session.

Declaration

Function GetTitle$ As String
Function GetTitle As Variant

Visual Basic

ttGetTitle
(**ByVal** *HSess* **As Long**,
ByVal *Title* **As String**,
ByVal *BuffSize* **As Long**)
As Long

C language

int ttGetTitle

(**HSess** *HSess*,
LPSTR *Title*,
unsigned *BuffSize*);

Remarks The title contents, without the "TTWIN - " prefix, are returned. The title is set using **ttSetTitle**.

Parameter *Title* is the title of the session.

BuffSize is the size of *Title*.

Return value Returns the title of the current TTWIN session.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
<0	The function failed. *Refer to Appendix B - Constants on page 381* for a complete list of return codes.

Menu Configure > Title

Example

```
Title$ = GetTitle$
```

The title contents, less the "TTWIN - " prefix, is returned via the string variable *Title$*.

ttGetTTWinPath$ ()

Description Gets the full path for the `TTWIN.EXE` file.

Declaration

```
Function GetTTWinPath$  As String
Function GetTTWinPath  As Variant
```

Visual Basic

ttGetTTWinPath
(**ByVal** *HSess* **As Long**,
ByVal *Path* **As String**,
ByVal *BuffSize* **As Long**)
As Long

C language

int ttGetTTWinPath
(**HSess** *HSess*,
LPSTR *Path*,
unsigned *BuffSize*);

Parameters

Path contains the full path name for the `TTWIN.EXE` file.

BuffSize is the size of*Path*

Return value

Returns the full path for the `TTWIN.EXE` file.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

The function returns the following valuesPath =

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

GetTTWinPath

Example

GoTo

Description Forces script execution to be transferred to the statement following the given label.

Syntax **GoTo** *Label*

Parameter *Label* is the line label for the transfer of execution.

Example

```
If Time = 12 Then GoTo Noon
…
Noon:
…
```

If the *Time* variable is equal to12 then script execution transfers to, and continues from, the line marked by the **GoTo** label, *Noon.*

GroupBox

Description Defines a group box.

Syntax **GroupBox** *x, y, cx, cy, Title$*

Remarks A group box brackets related items within an area on the dialog box. The box drawn can have a title, if required. Within a group box can be a combination of buttons, check boxes, combo boxes, list boxes, option buttons, text and text boxes.

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of the group box, relative to overall dialog box.

cx, cy are the width (*cx*) and height (*cy*) of the group box.

Title$ is the group box caption.

Example

```
GroupBox 8, 8, 62, 58,"Options"
CheckBox 2, 17, 50, 20,"Local Echo", intvar
CheckBox 2, 31, 50, 20,"&Autowrap", intvar
CheckBox 12, 45, 50, 20,"&Insert Mode", intvar
```

A group box with the caption **Options** will be created containing three check boxes: **Local Echo**, **Autowrap** and **Insert Mode**, see below.

See also **Button**, **CancelButton**, **CheckBox**, **ComboBox**, **ListBox**, **OKButton**, **OptionButton**, **TextBox**

ttHangup ()

Description Hangs up the modem.

Declaration **Sub Hangup**

Visual Basic **ttHangup** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttHangup** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Action > Hangup

Example

```
ModemInit$ = "ATZ" + Chr$(13)
...
SendData ModemInit$
If (WaitForString (6, OkResponse$) = 1) Then
                    ' Modem initialisation successful
     SendData DialPrefix$ + PhoneNumber$ + DialSuffix$
     Response = WaitForString (Timeout, "CONNECT",_
     "BUSY", NO CONNECT", "NO CARRIER")
     If (Response = 1) Then
      ' We received the first search string, "CONNECT"
          Successful = True
     End If
End If
If Successful Then
...
Hangup
...
```

The above example sends the initialisation string to the modem and waits for a response. If the connection is successful the conversation is continued until the modem is hungup.

Example

See also **ttDialEntry**, **ttDialManual**, **ttSelectModem**

See also **ttDial**, **ttDialEntry**, **ttDialManual**, **ttSelectModem**

Hex$ ()

Description Returns a hexadecimal value.

Declaration **Function Hex**$ (**ByVal** *Expression* **As Any**) **As String**

Remarks The number is rounded to the nearest whole number before evaluating.

Parameter *Expression* is any valid numeric or string expression.

Return value Returns a hexadecimal value of *Expression*.

Example
```
HexValue = Hex (59)
```
HexValue contains 3B.

See also **Oct$**

If...Then...Else

Description Allows the execution of statements based on a condition.

Syntax **If** *Expression* **Then** *Statement1(s)* [**Else** *Statement2(s)*]

Remarks In the single-line form of the **If** statement, the complete statement, including those statements conditionally executed, **must** be one line.

Parameters If the value of *Expression* is non zero, it is treated as TRUE and *Statement1(s)* is executed. If the value of *Expression* is zero, then *Statement2(s)*, if it exists, is executed.

Example

```
If State = 0 Then Display ("OFF") Else Display ("ON")
```

Assuming is always) or 1, this example will display either "OFF" when *State* = 0 or "ON" when *State* = 1.

See also **If...Then...ElseIf**

If...Then...ElseIf

Description Allows the execution of statements based on a condition.

Syntax

If *Expression* **Then**
    [*Statement(s)*]
[**ElseIf** *ElseIfExpression* **Then**
    [*ElseIfStatement(s)*]]
[**Else**
    [*ElseStatement(s)*]]
End If

Remarks The multi-line form of the **If** statement provides more flexibility than the single line form, with the possibility of multiple conditions.

Note: *Unlike the* ***If...Then...Else*** *the* ***If...Then...ElseIf*** *must end with an* ***End If****.*

Parameters If *Expression* is TRUE (non zero) then *Statement(s)* are executed.

If *Expression* is FALSE (zero) then the first *ElseIfExpression*, if it exists, is evaluated.

Each *ElseIfExpression* (if any) is evaluated in turn and if the *ElseIfExpression* is TRUE then the *ElseIfStatement(s)* following it is executed.

If ALL of the *ElseIfExpressions* are FALSE then the *ElseStatement(s)*, if any, are executed.

Example 1

```
Hello = 1
If Hello Then
     Display ("Hello")
End If
```

This will print the word "Hello" since the value of the expression is non zero (i.e., the value of the variable *Hello*).

In the case of an **ElseIf** statement, the operation is exactly the same as if a new **If** statement had been inserted.

Example 2

```
If (Day > 28) And (Leapyear=0) Then
     Display ("Too many days for February")
ElseIf  (Month <> 2) Then
     Display ("Wrong month for February")
Else
     Display ("Just right for February")
End If
```

In this case, it is assumed that the variables *Day, Leapyear* and *Month* have been set up prior to the statement being executed. The program segment prints one of three messages, based on the *Day, Leapyear* and *Month* variables.

See also **If...Then...Else**

Input$ ()

Description Returns characters from an open file.

Declaration **Function Input$** (**ByVal** *CharNum%* **As Integer**, [#]**ByVal** *FileNum%* **As Integer**) **As String**

Function Input (*CharNum%* **As Integer**, [#]*FileNum%* **As Integer**) **As Variant**

Remarks The file must be either a sequential or binary file and opened in either **Input** or **Binary** mode. The **Input$** function returns all characters including punctuation marks and spaces.

Parameters *CharNum%* is the number of characters to be read.

FileNum% is the file number.

Return value Returns *CharNum%* characters from the open file, *FileNum%*.

Example

```
FileNew = "C:\test.txt"
...
Open FileNew For Input As 1
While Not EOF(1)
InChars = Input(5, 1)
Wend
Close 1
```

ttInsertData ()

Description

Visual Basic

ttInsertData
(**ByVal** *HSess* **As Long**,
ByVal *Data* **As String**,
ByVal *Length* **As Long**)
As Long

C language

int ttInsertData
(**HSess** *HSess*,
LPCSTR *Data*,
unsigned *Length*);

Parameters

Data is the data to be sent to the host/displayed on the screen

Length is number of characters in the data string??????????????????//

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

InStr ()

Description Returns the position of a substring within a string

Declaration **Function InStr** ([**ByVal** ***Start%*** **As Integer**,] **ByVal** *SearchStr$* **As String**, **ByVal** *SubStr$* **As String**[, **ByVal** *Case%* **As Integer**]) **As Integer**

Parameters If *Start%* is specified, the search starts from this character position.

SearchStr$ is the string that will be searched.

SubStr$ is the substring to be searched for.

If *Case%* is specified and non zero, then the comparison is case insensitive. If *Case%* is specified and zero, the comparison is binary.

Return value The function return value is as follows:

Value	Meaning
0	The substring was not found.
>0	The start position of the first occurrence of the substring.

Example

```
InStr ("asfKjkltrokeit", "k", 1)
```

Returns 4, being the first occurrence of "k" in the main string, in a case insensitive search.

See also **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

Int ()

Description Returns the integer part of a number.

Declaration

Function Int (**ByVal** *Num* **As Double**) **As Double**
Function Int (**ByVal** *Num* **As Integer**) **As Integer**
Function Int (**ByVal** *Num* **As Long**) **As Long**
Function Int (**ByVal** *Num* **As Single**) **As Single**
Function Int (**ByVal** *Num* **As Variant**) **As Variant**

Parameter *Num* is any valid numeric expression.

Return value **Int** removes the fractional portion of a number, returning an integer, as follows:

Num	Returned integer
< 0	<= *Num*
>= 0	<= *Num*

Example 1

```
IntNumber = Int (91.2)
```

IntNumber contains 91.

Example 2

```
IntNumber = Fix (-79.12)
```

IntNumber contains -80.

Example 3

```
IntNumber = Fix (-56.73)
```

IntNumber contains -57.

See also **Fix**

ttIsConnected ()

Description Determines if there is a connection to a remote host.

Declaration **Function IsConnected As Integer**

Visual Basic **ttIsConnected** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttIsConnected** (**HSess** *HSess*);

Return value The possible return values are:

Value	Meaning
zero	Not connected.
non zero	Connected.

Return value The function returns the following values:

Value	Meaning
1	The function succeeded/connected.
0	Not connected.
< 0	The function failed/not connected. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
If Not IsConnected Then
    MsgBox "You must be connected and logged in to run_
this script", 48
End If
```

If the current seesion is not connected to a remote host a message box is displayed.

Example

See also **Break**, **Connect**, **Disconnect**,

See also **ttBreak**, **ttConnect**, **ttDisconnect**

ttIsDDEItemDefined ()

Description Determines if an DDE item exists.

Declaration **Function IsDDEItemDefined** (**ByVal** *Topic$* **As String**, **ByVal** *Item$* **As String**) **As Integer**

Visual Basic
ttIsDDEItem
(**ByVal** *HSess* **As Long**,
ByVal *Topic* **As String**,
ByVal *Item* **As String**)
As Long

C language
int ttIsDDEItem
(**HSess** *HSess*,
LPCSTR *Topic*,
LPCSTR *Item*);

Remarks That is, a that an item has been created using **CreateDDEItem** and that the item has not yet been destroyed with **DestroyDDEItem**.

Parameters *Topic$* is the subject of the conversation. (An example topic, for Microsoft Excel would be a `.XLS` file.)

Item$ is the actual item of data being passed. (An example item, for Microsoft Excel would be a cell reference, such as, R1C5.)

Return value The possible return values are as follows:

Value	Meaning
non zero	The specified DDE item exists (i.e., it has been previously defined and not yet destroyed).
zero	The specified DDE item does not exists.

Return value The function returns the following values?????????:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
ItemState = IsDDEIemDefined (Topic2$, Item3$)
```

Example

See also **CreateDDEItem**, **DestroyDDEItem**, **GetDDEItem**, **SetDDEItem**

See also **ttCreateDDEItem**, **ttDestroyDDEItem**, **ttGetDDEItem**, **ttSetDDEItem**

IsEmpty ()

Description Indicates whether a variable has been initialised.

Declaration **Function IsEmpty** (**ByVal** *Expression* **As Variant**) **As Integer**

Parameter *Expression* is any valid numeric or string expression of type **Variant**.

Return value The possible return values are:

Value	Meaning
non zero	*Expression* is uninitialised or has been set to **Empty**.
zero	*Expression* has been initialise or there is more than on variable in *Expression*.

See also **IsNull**, **IsNumeric**

IsNull ()

Description Indicates whether a variable contains valid data.

Declaration **Function IsNull** (**ByVal** *Expression* **As Variant**) **As Integer**

Parameter *Expression* is any valid numeric or string expression of type **Variant**.

Return value The possible return values are:

Value	Meaning
non zero	*Expression* is **Null**, i.e. it contains no valid data. This is also returned if any of the variables within *Expression* are **Null**.
zero	*Expression* contains valid data.

Example

```
Expression = Null
Valid = IsNull (Expression)
```

Valid is non zero.

See also **IsEmpty**, **IsNumeric**

IsNumeric ()

Description Indicates whether an expression is numeric when evaluated.

Declaration **Function IsNumeric** (**ByVal** *Expression* **As Variant**) **As Integer**

Parameter *Expression* is any valid numeric or string expression of type **Variant**.

Return value The possible return values are:

Value	Meaning
non zero	*Expression* is numeric.
zero	*Expression* is non numeric.

Example

```
a = 671.43
b = 53.0
Expression = IsNumeric (a + b)
```

See also **IsEmpty**, **IsNull**

ttKeyBoardLocked ()

Description Determines whether the keyboard is locked.

Declaration **Function KeyBoardLocked As Integer**

Visual Basic **ttKeyBoard Locked**
(**ByVal** *HSess* **As Long**)
As Long

C language **int ttKeyBoard Locked**
(**HSess** *HSess*);

Return value The possible return values are:

Value	Meaning
1	(True) Locked
0	(False) Not locked
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
If KeyBoardLocked Then ...
```

ttKeyText ()

Description

Visual Basic

ttKeyText
(**ByVal** *HSess* **As Long**,
ByVal *Text* **As String**)
As Long

C language

int ttKeyText
(**HSess** *HSess*,
LPCSTR *Text*);

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Parameter

Text is

Kill

Description Deletes a file from a disk.

Declaration **Sub Kill** (**ByVal** *Path$* **As String**)

Parameter *Path$* specifies one or more files to be deleted. The wildcards * and ? are supported. An error occurs if you try to delete an open file.

Example
```
Kill *.doc
```
All files with a `.DOC` extension will be deleted.

LBound ()

Description Returns the lower bound of the specified dimension for an array.

Declaration **Function LBound** (**ByVal** *Array* **As Any**[, **ByVal** *Dimension%* **As Integer**]) **As Integer**

Parameters *Array* is any valid array variable.

Dimension% indicates the dimension for which the lower bound is to be returned. Use 1 for the first dimension, 2 for the second, etc. If *Dimension%* is not specified, 1 is assumed.

Return value Returns the lower bound *Array* and *Dimension%* (if specified).

Example

```
Dim Data (-1 to 1, 25 to 100)
Lower1 = LBound (Data, 1)
Lower2 = LBound (Data, 2)
```

Lower1 contains -1 and *Lower2* contains 25.

See also **UBound**

LCase$ ()

Description Returns a string with every uppercase character converted to lowercase.

Declaration **Function LCase$** (**ByVal** *String$* **As String**) **As String**
Function LCase (**ByVal** *String$* **As String**) **As Variant**

Parameter *String$* is the string to be converted into lowercase.

Return value Returns *String$* as a lowercase string.

Example
```
LCase ("12abCD")
```
Returns "12abcd".

See also **UCase$**

Left$ ()

Description Returns the leftmost substring of a main string.

Declaration **Function Left$** (**ByVal** *String$* **As String**, **ByVal** *Length%* **As Integer**) **As String**
Function Left (**ByVal** *String$* **As String**, **ByVal** *Length%* **As Integer**) **As Variant**

Parameters *String$* is the main string.

Length% is the length, in characters, of the substring to be returned.

Return value Returns the leftmost *Length%* characters of *String$*. If *String$* is shorter than *Length%*, all of *String$* is returned.

Example
```
Left$ ("1234567890", 3)
```
Returns "123".

See also **InStr**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

Len ()

Description Returns the length of a string.

Declaration **Function Len** (**ByVal** *String$* **As String**) **As Integer**

Parameter *String$* is any valid string.

Return value Returns the length of *String$*.

Example
```
Len ("2468abcde")
```
Returns 9.

See also **InStr**, **Left$**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

Line Input

Description Reads a line from a file.

Syntax **Line Input** #**ByVal** *FileNum%, Input$*

Remarks The file is read one character at a time until a carriage return/linefeed sequence is found. (The carriage return/linefeed is not appended to the returned string.)

Parameter *FileNum%* is a valid file number of an open sequential file.

Input$ is the **String** variable to which the line is assigned.

Example

```
Open "c:\hold.txt" for input as 1
Line Input 1, input$
Close 1
```

ListBox

Description Defines a list box control.

Syntax **ListBox** *x, y, cx, cy, Options$, State%*

Remarks A list box displays a list of options in the order in which they are specified. If the space allocated for the list is too small then a scroll list is created.

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of the group box, relative to overall dialog box.

cx, *cy* are the width (*cx*) and height (*cy*) of the list box.

Options$ is a list of options to be placed in the list box. This a single string of tab separated fields. Each field is a line in the list box. If the list of options is too long for the space allocated then a scroll list is created.

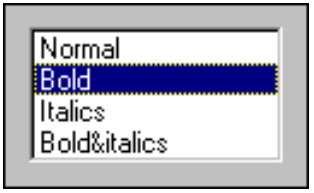
State% is an integer variable identifying the current selection. The variable is set as follows:

State%	Option selected
-1	No selection
0	First option
1	Second option
etc.	...

Example 1

```
State1 = 1
...
ListBox  40, 20, 70, 45, "Normal  Bold  Italics_
Bold&italics, State1
```

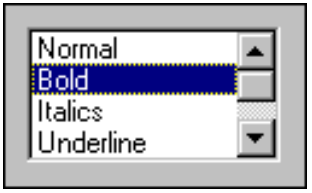
A list box is created containing the options: **Normal**, **Bold**, **Italics** and **Bold&italic**, see below.



Example 2

```
State1 = 1
...
ListBox  40, 20, 70, 45, "Normal  Bold  Italics_
Underline  Bold&italics  Bold&underline", State1
```

As the list box is too small, a scroll list is created containing the options: **Normal**, **Bold**, **Italics**, **Underline**, **Bold&italic** and **Bold&underline**, see below.



See also **CheckBox**, **ComboBox**, **GroupBox**, **OptionButton**, **TextBox**

LOF ()

Description Returns the size, in bytes, of an open file.

Declaration **Function LOF** (**ByVal** *FileNumber%* **As Integer**) **As Long**

Parameter *FileNumber%* is any valid file number. The file must first be opened using the **Open** statement.

Return value Returns the size, in bytes, of the open file, *FileNumber%*.

Example

```
Open MyFile As FileNo
Length = LOF (FileNo)
```

See also **EOF**

Log ()

Description Calculates the natural logarithm of a number.

Declaration **Function Log** (**ByVal** *Number* **As Double**) **As Double**
Function Log (**ByVal** *Number* **As Single**) **As Single**

Parameter *Number* is any valid numeric expression greater than zero.

Return value Returns the natural logarithm of *Number*.

Example
```
NaturalLog = Log (x)
```

LTrim$ ()

Description Returns a string with all leading spaces and tabs removed.

Declaration **Function LTrim$** (**ByVal** *String$* **As String**) **As String**
Function LTrim$ (**ByVal** *String$* **As Variant**) **As Variant**

Parameter *String$* is any valid string.

Return value Returns *String$* with all leading spaces and tabs removed.

Example
```
LTrim ("     hello")
```
Returns "hello".

See also **InStr**, **Left$**, **Len**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

Mid$ ()

Description Returns a substring of a given string.

Declaration **Function Mid$** (**ByVal** *String$* **As String**, **ByVal** *Index%* **As Integer**, [**ByVal** *Len%* **As Integer**]) **As String**

Function Mid (**ByVal** *String$* **As String**, **ByVal** *Index%* **As Integer**, [**ByVal** *Len%* **As Integer**]) **As Variant**

Parameters *String$* is the main string.

Index% is the starting point for the substring.

Len% is the length of the substring to be returned. If *Len%* is not given then a substring from the position specified by *Index%* to the end of *String$* is returned.

Return value Returns a substring of length *Len%*, of the main string *String$* starting from the character position *Index%*.

Example 1

```
Mid$ ("Hello there Fred", 7, 5)
```

Returns "there".

Example 2

```
Mid$ ("Hello there Fred", 7)
```

Returns "there Fred"

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

MilliPause

Description Pauses script execution for milliseconds.

Declaration **Sub MilliPause** (**ByVal** *Sec%* **As Integer**)

Parameter Script execution pauses for *Sec%* milliseconds.

Example
```
MilliPause 5
```
Script execution will pause for 5 milliseconds.

See also **Pause**

MkDir

Description Creates a new directory.

Declaration **Sub MkDir** (*Pathname$* **As String**)

Parameter *Pathname$* is any valid string identifying a path. If a drive is not specified, the current drive is assumed.

Example
```
MkDir d:\templates
```
Makes the new directory `TEMPLATES` on the `D:` drive.

See also **ChDir**, **ChDrive**, **RmDir**

ttMoveCursor ()

Description Moves the cursor to nominated Row, Col, only available for selected emulations.

Visual Basic

ttMoveCursor
(**ByVal** *HSess* **As Long**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**)
As Long

C language

int ttMoveCursor
(**HSess** *HSess*,
int *Row*,
int *Col*);

Parameter The cursor is moved to row and column values in *Row* and *Col* respectively. The top row and leftmost column both have a value of 1.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

MsgBox

Description Displays a message box.

Declaration **Sub MsgBox** (**ByVal** *Message$* **As String**[, **ByVal** *Type%* **As Integer**][, **ByVal** *Title$* **As String**])

Remarks Because the **MsgBox** subroutine does not return a value, the use of buttons other than the **OK** button is not recommended. If you want to utilize other buttons, use the **MsgBox** function, see *page 272*.

Message boxes can also be a useful debugging tool.

Parameters *Message$* is the message to appear within the message box.

Type% specifies the symbols and buttons to be displayed in the box. It is the sum of values from the following groups:

Style	Value	Item displayed
Buttons	0	OK button (default)
	1	OK and cancel buttons
	2	Abort, retry and ignore buttons
	3	Yes, no and cancel buttons
	4	Yes and no buttons
	5	Retry and cancel buttons
Icons	0	No icon (default)
	16	Stop icon
	32	Question icon
	48	Attention icon
	64	Information icon

Style	Value	Item displayed
Button action	0	First button is the default
	256	Second button is the default
	512	Third button is the default

If *Type%* is omitted, the default is 0.

Title$* is the title of the message box. If *Title$ is omitted, the title defaults to the name of the application.

Example 1

```
EHeading$ = "Logon Error"    ' Heading for error windows
MsgBox "Login aborted after three" + Chr$(13) + "_
unsuccessful attempts", 16, EHeading$
```

The message box has a stop icon (16) and an OK button, .

MsgBox ()

Description Displays a message box and returns the button selected.

Declaration **Function MsgBox** (**ByVal** *Message$* **As String**[, **ByVal** *Type%* **As Integer**][, **ByVal** *Title$* **As String**]) **As Integer**

Remarks Message boxes can also be a useful debugging tool. (There is also a subroutine form of **MsgBox** but it is only recommended for use with an **OK** button, see *page 270*.)

Parameters *Message$* is the message to appear within the message box.

Type% specifies the icons and buttons to be displayed in the box and is the sum of values from the following groups:

Style	Value	Item displayed
Buttons	0	OK button (default)
	1	OK and cancel buttons
	2	Abort, retry and ignore buttons
	3	Yes, no and cancel buttons
	4	Yes and no buttons
	5	Retry and cancel buttons
Icons	0	No icon (default)
	16	Stop icon
	32	Question icon
	48	Attention icon
	64	Information icon
Button action	0	First button is the default
	256	Second button is the default
	512	Third button is the default

If *Type%* is omitted, the default is 0.

Title$* is the title of the message box. If *Title$ is omitted, the title defaults to the name of the application.

Return value The possible return values are:

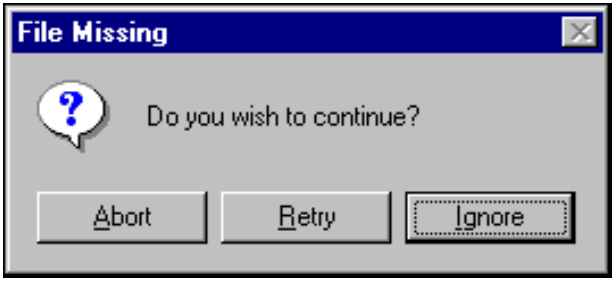
Value	Button selected
1	OK button
2	Cancel button
3	Abort button
4	Retry button
5	Ignore button
6	Yes button
7	No button

Example 1

```
IntVar = MsgBox ("Do you wish to continue?", 546, "File Missing")
```

A message box with title of "File Missing", containing the message "Do you wish to continue?" is displayed.

The box has abort , retry and ignore buttons, and a question icon with the third (ignore) button as the default (2 + 32 + 512 = 546).



Name

Description Renames a file.

Syntax **Name** *OldName$* **As** *NewName$*

Remarks Both new name and old name **must** be on the same drive, and the file specified in *OldName$* **must** be closed.

Wildcards, (*) and (?), are not supported.

Using **Name**, you can move a file from one directory or folder to another, but you **cannot** move a directory or folder.

Parameter *OldName$* is the current name and location. This can include the directory or folder, and drive.

NewName$ is the new name and location. The file name specified by must not already exist.

Example

```
Name text.txt As data.txt
```

Oct$ ()

Description Returns an octal value.

Declaration **Function Oct$** (**ByVal** *Expression* **As Any**) **As String**
Function Oct (**ByVal** *Expression* **As Any**) **As Variant**

Remarks The number is rounded to the nearest whole number before evaluating.

Parameter *Expression* is any valid numeric or string expression.

Return value Returns *Expression* as an octal value.

Example
```
OctValue = Oct (64)
```
OctValue contains 100.

See also **Hex$**

OKButton

Description Defines an OK button.

Syntax **OKButton** *x, y, cx, cy*

Remarks The dialog box closes when the OK button is pressed. If there is a **ButtonGroup** statement, the specified variable will be set to 0.

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of button, relative to dialog box being configured.

cx, cy are the width (*cx*) and height (*cy*) of the button.

Example
```
OKButton  40, 53, 36, 24
```
An OK button will appear in the dialog box.

See also **Button**, **ButtonGroup**, **CancelButton**, **OptionButton**

Open

Description Opens a file for input and output.

Declaration **Sub Open** *Path$* **As String** [**For** *Mode*] [**Access** *Access*] [*Lock*] **As** [#]**ByVal** *FileNumber%* **As Integer** [**Len**=*Length*]

Parameters *Path$* is a string that specifies the filename. The path name can include the directory and drive.

Mode specifies **Append**, **Binary**, **Input**, **Output** or **Random**.

Access specifies the operations to be permitted: **Read**, **Write** or **Read Write**.

Lock specifies the operations permitted on the open file by other processes: **Lock Read**, **Lock Write**, **Lock Read Write**, **Shared**.

FileNumber% is a valid file number obtained using **FreeFile**.

With files opened for random access, *Length* is the record length. With sequential files, *Length* is the number of characters buffered. *Length* is ignored when *Mode* is **Binary**.

Example

```
FileNew = "C:\test.txt"
Open FileNew For Input As 1
```

See also **Close**, **FileAttr**, **FreeFile**

OpenHostStream

Description Opens the host stream for reading.

Declaration **Sub OpenHostStream**

Remarks This function **MUST** be called before using **ReadHost$** or **ReadHostLine$**. After a **OpenHostStream** call, all data received from the host is placed into a queue for use by scripts (using **ReadHost$** or **ReadHostLine$**). Once the script has finished reading, the data stream **MUST** be closed. See **CloseHostStream**.

Example

```
Dim DataStr$ as String
...
OpenHostStream
     DataStr$ = ReadHostLine$
     ...
CloseHostStream
```

See also **CloseHostStream**, **ReadHost$**, **ReadHostLine$**

ttOpenInputCapture ()

Description Opens the host data stream for reading.

Visual Basic **ttOpenInputCapture** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttOpenInputCapture** (**HSess** *HSess*);

Remarks This **must** be called before a **ReadInputCapture**.

After an **OpenInputCapture** call, all data received from the host is placed into a queue for use by scripts (using **ReadInputCapture**). Once the script has finished reading, the data stream **MUST** be closed.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttCloseInputCapture**, **ttReadInputCapture**

OptionButton

Description Defines an option button

Syntax **OptionButton** *x, y, cx, cy, Option$*

Remarks Only one option button in a group of option buttons is can be selected at any one time.

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of button, relative to dialog box being configured.

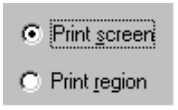
cx, cy are the width (*cx*) and height (*cy*) of the button.

Option$ is the text that is to appear on the button. An & preceding a character within the text string will produce an ALT shortcut key. For example, "&Echo" will generate **Echo**. This enables **Echo** to be selected using the key sequence, ALT_E.

An **OptionGroup** statement must appear before an **OptionButton** statement.

Example

```
ButSelect = 0
OptionGroup ButSelect
OptionButton  12, 31, 50, 20, "Print &screen"
OptionButton  22, 31, 50, 20, "Print &region"
```



The two options, Print screen and Print region are displayed with the first option the default (*ButSelect* = 0).

See also **CheckBox**, **ComboBox**, **GroupBox**, **ListBox**, **OptionGroup**

Option Explicit

Description This statement forces variables that are used in TTWIN BASIC programs to be explicitly defined.

Syntax **Option Explicit**

Remarks By default, variables are created and initialised the first time they are used. While this can be very useful for quick program writing, it can then be difficult to locate errors due variable names being misspelt.

Example

```
Option Explicit
Dim IntVar1
...
IntVar1 = IntVa1 + 1                    'spelling mistake
...
```

By using the **Option Explicit** in the above example, the misspelt integer variable, *IntVar1*, will be easily detected when the code is verified or when an attempt is made to run it.

OptionGroup

Description Defines the start of a new group of **OptionButton** controls.

Syntax **OptionGroup** *Button%*

Remarks An **OptionGroup** statement **MUST** appear before an **OptionButton** statement or group of **OptionGroup** statements. When one option button in a group is selected all others in the same group are deselected.

There can be more than one **OptionGroup** statement in a dialog box.

Parameter The variable *Button%* indicates which option button in the group is selected, as follows:

Button%	Button selected
0	First button
1	Second button
2	Third button
etc.	...

Example

```
GroupBox 8, 8, 62, 58, "Options:"
     OptionGroup intvar
     OptionButton 12, 17, 50, 20, "Local Echo"
     OptionButton 12, 31, 50, 20, "&Autowrap"
     OptionButton 12, 45, 50, 20, "&Insert Mode"
     intvar = 2
```

A group box is displayed with the options, Local Echo, Autowrap and Insert Mode, with the third option selected (*intvar* = 2).

See also **OptionButton**

Pause ()

Description Pauses script execution.

Declaration **Sub Pause** (**ByVal** *Period%* **As Integer**)

Visual Basic **ttPause**
ByVal *Period* **As Long**)
As Long

C language **Void ttPause**
UINT *Period*);

Parameter Script execution pauses for *Period%* seconds.

Example Pause 5
Script execution will pause for 5 seconds.

Example

See also **MilliPause**

ttPrintClose ()

Description Closes all active print jobs.

Declaration **Sub PrintClose**

Visual Basic **ttPrintClose** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttPrintClose** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > Close print run

Example `PrintClose`

Example

See also **PrintScreen**, **PrintSelected**

See also **ttPrintScreen**, **ttPrintSelected**

ttPrintScreen ()

Description Prints the entire emulation screen region.

Declaration **Sub PrintScreen** ([**ByVal** *Mode%* **As Integer**])

Visual Basic **ttPrintScreen**
(**ByVal** *HSess* **As Long**,
ByVal *Mode* **As Long**)
As Long

C language **int ttPrintScreen**
(**HSess** *HSess*,

int *Mode*);

Parameter *Mode%* specifies the printing mode. The possible values for *Mode%* are:

Mode%	Printing mode
0	Currently configured mode (Raw or Windows).
1	Raw print.
2	Windows printing.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu File > Print...

Example

```
PrintScreen
```

Example

See also **PrintClose**, **PrintSelected**

See also **ttPrintClose**, **ttPrintSelected**

ttPrintSelected ()

Description Prints the selected region.

Declaration **Sub PrintSelected** ([**ByVal** *Mode%* **As Integer**])

Visual Basic

ttPrintSelected
(**ByVal** *HSess* **As Long**,
ByVal *Mode* **As Long**)
As Long

C language

int ttPrintSelected
(**HSess** *HSess*,
int *Mode*);

Parameter

Mode% specifies the printing mode. The possible values for *Mode%* are:

Mode%	Printing mode
0	Currently configured mode (Raw or Windows).
1	Raw print.
2	Windows printing.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu

File > Print...

Example

```
DispSelectCont 1,2,15,20
PrintSelected
```

A continuous region from row 1, column 2 to row 15, column 20 is selected (highlight) and then printed.

Example

See also

PrintClose, **PrintScreen**

See also **ttPrintClose**, **ttPrintScreen**

Randomize

Description Initialises the random number generator for the **Rnd** function.

Declaration **Sub Randomize** ([**ByVal** *Seed* **As Long**])

Parameter If *Seed* is omitted, the value returned by the system timer is used.

Example
```
Randomize
RandValue = Int (9 * Rnd) + 1
```
RandValue contains a value less than 10 and, greater than or equal 1.

See also **Rnd**

ReadHost$ ()

Description Reads all available data from the host queue.

Declaration **Function ReadHost$ As String**
Function ReadHost As Variant

Remarks **OpenHostStream** must be called before **ReadHost$**.

Return value Returns a string containing the host data read.

Example
```
Dim Data$ as String
...
```

```
OpenHostStream
      Data$ = ReadHost$
CloseHostStream
```

See also **CloseHostStream**, **OpenHostStream**, **ReadHostLine$**

ReadHostLine$ ()

Description Reads data from the host.

Declaration **Function ReadHostLine$** ([**ByVal** *Timeout%* **As Integer**,] [**ByVal** *TermChars$* **As String**]) **As String**

Remarks **OpenHostStream** must be called before **ReadHostLine$**.

Parameters If *Timeout%* is given as 0 or not given at all then the time out period is infinite.

If *TermChars$* is not given, carriage return is used as a terminator.

The host data is read until a character from *TermChars$* is received or until *Timeout%* has expired.

Return value Returns a string containing the host data read.

Example

```
Dim HostData$ as String
...
OpenHostStream
      HostData$ = ReadHostLine$ 10, Chr$(13)
CloseHostStream
```

See also **OpenHostStream**, **CloseHostStream**, **ReadHost$**

ttReadInputCapture ()

Description Reads the input dasta stream from the host.

ttReadInputCapture
(**ByVal** *HSess* **As Long**
(**ByVal** *Data* **As String**
(**ByVal** *Length* **As Long**)
As Long

C language

int ttReadInputCapture
(**HSess** *HSess*
(**LPSTR** *Data*
(**unsigned** *Length*);

Remarks After an **OpenInputCapture** call, all data received from the host is placed into a queue for use by scripts (using **ReadInputCapture**). Once the script has finished reading, it **MUST** be closed.

Return value The function returns the following values:

Value	Meaning
>=0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttCloseInputCapture**, **ttOpenInputCapture**

ReDim

Description Declares variables and arrays at procedure level, and allocates or reallocates the required memory space.

Syntax **ReDim** *Variable* [([*LowerBound* **To**] *UpperBound*)] [**As** *Type*]…

Remarks You can use **ReDim** to resize a dynamic array that has been formally declared. You can not use **ReDim** to change the type of the array unless the array in contained in a **Variant**.

Note: *The contents of the array will be lost.*

Parameters *Variable* is the name of the variable being declared.

The *LowerBound*, if used is the number of the first element in the array. That is, if the *LowerBound* is 3, then the first array element is 3 and there are no elements 1 or 2.

When *LowerBound* is not specified, a default of 0 is used.

The *UpperBound* is the number of elements in the array only when *LowerBound* is 1.

The **ReDim** statement can change the number of elements as well as the *LowerBound* and *UpperBound* values for each dimension. However, the number of dimensions in the array cannot change.

Arrays may be multi-dimensional, the total number of array elements is limited only by memory. All elements must live in one 64K memory block.

Type is one of the supported data types. When *Type* is not specified, **Integer** is assumed.

Example

```
Dim SmallArray (1 To 15)
....
ReDim SmallArray (6)
```

This changes the size of an already declared array called *SmallArray* to be 7 integer elements in size, the first element is 0, the last element is 6.

See also **Dim**

ttRegisterEventProc () C only

Description **Registration of the Event Handler**

C language

int ttRegisterEventProc
(**HSess** *HSess*,
int *EventID*,
FARPROC *Proc*);

Parameters

EventID is an event when user proc must be called.

Proc is user defined handler.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Return value The following table describes possible values of EventID and declarations of user defined handler:

Event ID	Value	PROC declaration sample
EVID_COMMOUT	0	LPSTR MY PROC (LPSTR Data, unsigned size)
EVID_COMMIN	1	
EVID_LMOUSEDOWN	2	VOID My PROC (int x, int y, int row, int col, int shifts)
EVID_LMOUSEUP	3	
EVID_RMOUSEDOWN	4	
EVID_RMOUSEUP	5	
EVID_MMOUSEDOWN	6	
EVID_MMOUSEUP	7	
EVID_MOUSEMOVE	8	

Rem

Description Indicates that the remainder of the line is a comment.

Syntax **Rem** *Comments*

Remarks If the **Rem** statement is used on a line already containing a TTWIN BASIC statement then the **Rem** must be preceded by a : (colon). TTWIN BASIC will ignore any text following this keyword.

Example

```
x = 39     Rem  This is comment #2
```

See also **‘** (apostrophe)

Replace$ ()

Description Replaces all occurrences of a character within a string with a new character.

Declaration **Function Replace$** (*String$* **As String**, **ByVal** *OldChar%* **As Integer**, **ByVal** *NewChar%* **As Integer**) **As String**

Function Replace (*String$* **As String**, *OldChar%* **As Integer**, *NewChar%* **As Integer**) **As Variant**

Parameters *String$* is the string to be altered.

OldChar% is the character (ASCII) in *String$* to be changed.

NewChar% is the character (ASCII) to replace *OldChar%*.

Return value. Returns a string containing *NewChar%* in place of *OldChar%*.

Example

```
Replace$ ("ABCABCABCABC", 65, 90)
```

Returns "ZBCZBCZBCZBC".

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Right$**, **RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

Reset

Description Closes all disk files opened with the **Open** statement.

Declaration **Sub Reset**

Example

```
Reset
```

See also **Close**, **End**, **Open**

Resume

Description Resumes execution of TTWIN BASIC script after an error handling routine.

Syntax 1 **Resume** [0]

Syntax 2 **Resume Next**

Syntax 3 **Resume** *Label*

Remarks The **Resume** statement has 3 forms:

Resume [0]

Execution resumes with the statement that caused the error.

Resume Next

Execution resumes with the statement immediately following the statement that caused the error.

Resume *Label*

Execution resumes at the line specified by *Label*. *Label* is a label or line number, and **must** be in the same procedure as the error handler.

Note: *A* ***Resume*** *statement* ***must*** *be included in an error handling routine. However, if you use a* ***Resume*** *statement anywhere except in an error-handling routine, an error occurs.*

Example

```
Sub Example(Var1, Var2)
     On Error GoTo ErrorHandler
                         'Enable error-handling routine.
     ...
     ...
     Exit Sub                'No error - do not execute
                                    'the error handler
     ErrorHandler:             ' Error-handling routine
          Select Case Err      ' Evaluate error number
          Case -1
               ...
          Case -2, -5
               ...
          Case Else
               ...
          End Select
          Resume Next
       ' Resume execution with the statment immediately
            'following the line that caused the error
End Sub
```

In the above example, if an error occurs the execution jumps to the error handler indicated by the label, *ErrorHandler*. Within the error handler, a **Select Case** statement is then

used to resolve the error. Execution is then returned, with the **Resume Next** statement, to statement after to one that caused the error.

See also Refer to *Chapter 8 - Run-time Error Handling on page 77*.

Right$ ()

Description Returns the rightmost characters of a string.

Declaration **Function Right$** (**ByVal** *String$* **As String**, **ByVal** *Len%* **As Integer**) **As String**

Function Right (**ByVal** *String$* **As String**, **ByVal** *Len%* **As Integer**) **As Variant**

Parameters *String$* is the main string.

Length% is the length, in characters, of the substring to be returned.

Return value Returns the rightmost *Length%* characters of *String$*. If *String$* is shorter than *Length%*, all of *String$* is returned.

Example

```
Name = Right$ ("Hello there Fred", 4)
```

Name contains "Fred".

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, RTrim$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

RmDir

Description Removes a directory.

Declaration **Sub RmDir** (*Directory$* **As String**)

Parameter *Directory$* is the path name of the directory to be removed. If the drive is not specified, the directory on the current drive is removed. If the directory is not empty, an error occurs. Use **Kill** to delete all files first.

Example

```
Directory$ = "C:\User"
RmDir Directory$
```

See also **ChDir**, **ChDrive**, **Kill**, **MkDir**

Rnd ()

Description Returns a random number.

Declaration **Function Rnd As Single**

Remarks Before calling **Rnd**, use **Randomize** to initialise the random number generator.

Return value Returns a number less than 1 and, greater than or equal to 0.

Example

```
RandNo = Rnd
```

See also **Randomize**

RTrim$ ()

Description Returns a string with all the trailing spaces and tabs removed.

Declaration
Function RTrim$ (ByVal *String$* **As String**) **As String**
Function RTrim (ByVal *String$* **As String**) **As Variant**

Parameter *String$* is any valid string.

Return value Returns *String$* with all the trailing spaces and tabs removed.

Example
```
RTrim$ ("Hello there      ")
```
Returns "Hello there".

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **Space$**, **StrComp**, **String$, Trim$**, **Val**

ttRunScript ()

Description Runs the specified script.

Visual Basic
ttRunScript
(**ByVal** *HSess* **As Long**,
ByVal *Script* **As String**)
As Long

C language
int ttRunScript
(**HSess** *HSess*,
LPSTR *Script*);

Parameter *Script* is the path name location of the script to be executed

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

ttSearchField ()

Description Search for a field with particular attributes.

Note: This may not be available for all emulations.

Visual Basic

ttSearchField
(**ByVal** *HSess* **AsLong**,
ByVal *StartRow* **As Long**,
ByVal *StartCol* **As Long**,
ByVal *FieldAttr* **As Long**,
ByVal *Direction* **As Long**,
ByVal *Exact* **As Long**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**)
As Long

C language

int ttSearchField
(**HSess** *HSess*,
int *StartRow*,
int *StartCol*,
unsigned *FieldAttr*,
int *Direction*,
int *Exact*,
int FAR **Row*,
int FAR **Col*);

Parameter

StartRow and *StartCol* are the starting position of a search for a field whcih either exactly matches *FieldAttr* or contains the attributes that *FieldAttr* has.

Direction determines a forward search (value>= 0) or a backward search (value <0).

Exact - if set to 1 then an exact match of attributes is done, otherwise a match is determined to be all of the attributes in *FieldAttr* being in the field.

Row and *Col* returned giving their location of the located field.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

Seek

Description

Sets the position in a open file for the next file operation.

Declaration

Sub Seek ([#]**ByVal** *FileNumber* **As Integer**, **ByVal** *Position* **As Integer**)

Remarks

Performing a write operation after a **Seek** beyond the end of a file extends the file.

Parameters

FileNumber% is a valid file number obtained using **FreeFile**.

Position is a number from 1 to 2,147,483,647, inclusive, indicating where the next read or write is to occur. For files opened in **Random** mode, *Position* is the next record. For files opened in other modes, *Position* is the byte position.

Example

```
FileNew = "c:\data.txt"
Open FileNew For Input As 1
Byte = 3
While Not EOF(1)
     Seek 1, Byte
     InChars = Input(5, 1)
     Byte = Byte + 3
Wend
Close 1
```

See also **FreeFile**, **Open**

Select Case

Description Executes one of several groups of statements, based on one or more conditions.

Syntax

Select Case *TestExp*
[**Case** *ConditionList*
    [*Statement(s)*]] ...
[**Case Else**
    [*ElseStatement(s)*]]
End Select

Parameters *TestExp*, a numeric or string expression, is the condition tested.

ConditionList is a list of conditions to be matched. This can be any or all of the following:

- a **To** expression, e.g, **Case** 5 **To** 7
- an **Is** comparison, e.g., **Case** *ItemA* **Is** *ItemB*
- a group of expressions each separated by comma, e.g., **Case** 6, 7, 8 **To** 11
- a comparison e.g., **Case Is** > 3.

Each *ConditionList*, if any, is evaluated in turn. If *TestExp* matches an item in *ConditionList* then the *Statement(s)* following it are executed.

If none of the items in any of the **Case** *ConditionList* statements match *TestExp* then the *ElseStatement(s)*, if any, are executed.

Example

```
Test = "John"
Select Case Test
Case "James" To "Jeff"
     ...
Case "John"
     ...
Case Else
End Select
```

In the above example, the statements after the **Case** "John" statement will be executed.

SelectDefaultSession

Description Finds the default session.

Declaration **Sub SelectDefaultSession**

Example
```
SelectDefaultSession
Name$ = GetSessionFile$
```
Name$ will contain the full path for the DEFALUT.TWC.

See also **SelectHost**

ttSelectHost ()

Description Sets the Host Address for communication module.

Declaration **Sub SelectHost** (**ByVal** *Host* **As String**)

Visual Basic
ttSelectHost
(**ByVal** *HSess* **As Long**,
ByVal *Host* **As String**)
As Long

C language

int ttSelectHost
(**HSess** *HSess*,
LPCSTR *Host*);

Parameter

Host is the IP address of the host or host name.

Return value

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

Example

See also **SelectDefaultSession**

See also

ttSelectLightPenField ()

Description

IBM3270 use only - a light pen selectable field.

Visual Basic

ttSelectLightPenField
(**ByVal** *HSess* **As Long**,
ByVal *Row* **As Long**,
ByVal *Col* **As Long**)
As Long

C language

int ttSelectLightPenField
(**HSess** *HSess*,
int *Row*,
int *Col*);

Parameter

Row and *Col* identify the field that is to be selected. This assumes that the field is an IBM3270 light.

The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

ttSelectModem ()

Description Selects a predefined modem entry as the current modem.

Declaration **Sub SelectModem** (**ByVal** *ModemName$* **As String**)

Visual Basic

ttSelectHost
(**ByVal** *HSess* **As Long**,
ByVal *ModemName* **As String**)
As Long

C language

int ttSelectHost
(**HSess** *HSess*,
LPCSTR *ModemName*);

Parameter *ModemName$* is the name of the predefined modem.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
MdmType = "HAYES COMPATIBLE"
SelectModem (MdmType)
```

Example

See also **DialEntry**, **DialManual**, **Hangup**

See also **ttDial**, **ttDialEntry**, **ttDialManual**, **ttHangup**

SelectSessionByID

Description Selects a TTWIN session using the session ID.

Declaration **Sub SelectSessionByID** (**ByVal** *IDName$* **As String**)

Remarks The session ID is set using the **/l** switch when starting TTWIN.

Parameter *IDName$* is the session ID.

Example Given that a TTWIN session is started as follows:

```
c:\program files\ttwin\bin\ttwin.exe /lsession1
```

then,

```
ID$ = "session1"
t = GetSessionCount
If t <> 0 Then
     SelectSessionByID (ID$)
End If
```

selects this session.

See also **SelectSessionByTitle**

SelectSessionByTitle

Description Selects a TTWIN session using the session title.

Declaration **Sub SelectSessionByTitle** (**ByVal** *Title$* **As String**)

Remarks The title is set using **SetTitle**.

Parameter *Title$* is the title of the session.

Example

```
SetTitle "AS400"
...
Title$ = "AS400"
t = GetSessionCount
If t <> 0 Then
     SelectSessionByTitle (Title$)
End If
```

See also **SelectSessionByID**

SendData

Description Sends data out the comms line to the host.

Declaration **Sub SendData** (**ByVal** *DataStr$* **As String**)

Parameter *DataStr$* is the character string sent to the host.

Example

```
Password = "ahson7jf"
SendData Password
```

ttSendData ()

Description Sends data out the comms line to the host.

Visual Basic

ttSendData
(**ByVal** *HSess* **As Long**,
ByVal *Data* **As String,**
ByVal *Length* **As Long**)
As Integer

C language

int ttSendData
(**HSess** *HSess*,
LPCSTR *Data*,
unsigned *Length*);‘

Parameters *Data* is a character string to be sent to the host.

Length is the length of the buffer containing the data.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttSendString**.

SendKeys

Description Sends keystrokes to the currently active window as if typed at the keyboard.

Declaration **Sub SendKeys** (**ByVal** *Keys$* **As String**, **ByVal** *Wait%* **As Integer**)

Parameter *Wait%* indicates whether the control is to be returned immediately, as follows

Wait%	
0	(True) The keystrokes are processed before control is returned to the procedure.
1	(False) Control is returned to the procedure immediately.

Keys$ is the keystroke or keystorkes to be sent to the window.

A single keyboard character is specified by the character itself e.g. the letter G is respresented by the string "G". To specify a group of characters, append each character to the one preceding it. So for example, to represent the letters D, F, and Z, use the string "DFZ".

Special characters, such as, the percent sign (%) have special meanings to **SendKeys**. To specify one of these characters, enclose it within braces ({}), e.g, {%}.

To specify keys that aren't displayed, such as ENTER, use the codes shown below:

Key	Code
BACKSPACE	{BACKSPACE}, {BS}, or {BKSP}

Key	Code
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL OR DELETE	{DELETE} or {DEL}
DOWN, LEFT, RIGHT AND UP ARROW	{DOWN}, {LEFT}, {RIGHT} and {UP}
END	{END}
ENTER	{ENTER}or ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS OR INSERT	{INSERT} or {INS}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
F1, F2...F16	{F1}, {F2}...{F16}

Any key combinations with the SHIFT, CTRL, and ALT keys are specifed by preceding the key code with one or more of the following codes:

Key	Code
SHIFT	+
CTRL	^
ALT	%

To specify that the key should be held down while several other keys are pressed, enclose the code for those keys in parentheses. For example, to specify to hold down CTRL while A and X are pressed, use "^(AX)". To specify to hold down CTRL while A is pressed, followed by C without CTRL, use "^AX".

Example `SendKeys "Test^(AX) 0"`

See also **Display**

ttSendString ()

Description Sends data out the comms line to the host.

Visual Basic

ttSendString
(**ByVal** *HSess* **As Long**,
ByVal *Data* **As String**)
As Long

C language

int ttSendString
(**HSess** *HSess*,
LPCSTR *Data*);

Parameter *Data* is a pointer to a buffer containing the data.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also **ttSendData**.

ttSetCursorStyle ()

Description Changes the appearance of the cursor.

Declaration **Sub SetCursorStyle** (**ByVal** *Style%* **As Integer**)

Visual Basic
ttSetCursorStyle
(**ByVal** *HSess* **As Long**,
ByVal *Style* **As Long**)
As Long

C language
int ttSetCursorStyle (
HSess *HSess*,
unsigned *Style*);

Parameter *Style%* is a combination of:

Style%	Cursor style
&h00	Leave the cursor unchanged. (The default is cursor on and blinking.)
&h01	Full block cursor.
&h02	Small block cursor.
&h03	Horizontal line cursor.
&h04	Vertical line cursor.
&h40	Not visible, (visible otherwise).
&h80	Steady cursor, (blinking otherwise).

These values are delcared as constants in the header files; `TWCONST.H` and `TWDECL.BAS`. Refer to *Appendix B - Constants on page 381* for a complete list of declared constants.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Style% = &h84
SetCursorStyle Style%
```

Sets the cursor to appear as a vertical non-blinking line.

Example

See also **GetCursorStyle**

See also **ttGetCursorStyle**

ttSetDDEItem ()

Description Sets the value of a previously defined DDE item.

Declaration **Sub SetDDEItem** (**ByVal** *Topic$* **As String**, **ByVal** *Item$* **As String**, **ByVal** *Value$* **As String**)

Visual Basic

ttSetDDEItem
(**ByVal** *HSess* **As Long**,
ByVal *Topic* **As String**)
ByVal *Item* **As String**)
ByVal *Value* **As String**)
ByVal *Length* **As String**)
As Long

C language

int ttSetDDEItem
(**HSess** *HSess*,
LPCSTR *Topic*,
LPCSTR *Item*,
LPCSTR *Value*,
unsigned *Length*);

Parameters *Topic$* is the subject of the conversation. (An example topic, for Microsoft Excel would be a `.XLS` file.)

Item$ is the actual item of data being passed. (An example item, for Microsoft Excel would be a cell reference, such as, R1.)

Value$ the value of the item.

Length is the number bytes.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
Topic1$ = "System"
Item$ = "Path"
StartDDEServer
CreateDDEitem Topic1$, Item$
SetDDEitem Topic1$, Item$, GetTTWinPath
```

Example

See also **CreateDDEItem**, **DestroyDDEItem**, **SetDDEItem**, **IsDDEItem**

See also **ttCreateDDEItem**, **ttDestroyDDEItem**, **ttSetDDEItem**, **ttIsDDEItem**

ttSetDDEServerName ()

Description Sets the DDE server name for the current session of TTWIN.

Declaration **Sub SetDDEServerName** (**ByVal** *NewName* **As String**)

Visual Basic **ttSetDDEServerName**
(**ByVal** *HSess* **As Long**,
ByVal *NewName* **As String**)
As Long

C language **int ttSetDDEServerName**
(**HSess** *HSess*,
LPCSTR *NewName*);

Remarks There is a built-in default service called TTWIN but if you have more than one TTWIN session running you may need to rename the service.

Parameter *NewName* is the name of the DDE service.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example
```
SetDDEServerName "IBM"
```

Example

See also **StartDDEServer**, **StopDDEServer**

See also **ttStartDDEServer**, **ttStopDDEServer**

ttSetHostID ()

Description Assigns an ID to the host.

Visual Basic **ttSetHostID**
(**ByVal** *HSess* **As Long**,
ByVal *HostID* **As String**)
As Long

C language **int ttSetHostID**
(**HSess** *HSess*,
LPCSTR *HostID*);

Parameters *HostID* is the ID to be assigned to the host.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

See also

ttSetMenuMode ()

Description Changes the TTWIN main menu display.

Declaration **Sub SetMenuMode** (**ByVal** *State%* **As Integer**, **ByVal** *InSystem%* **As Integer**)

Visual Basic **ttSetMenuMode**

(**ByVal** *HSess* **As Long**,
ByVal *State* **As Long,**
ByVal *InSystem* **As Long**)
As Long

C language

int ttSetMenuMode
(**HSess** *HSess*,
int *State*,
BOOL *InSystem*);

Parameters

State% indicates the level at which the TTWIN main menu will be available, as follows:

State%	**Menu mode**
0	Full menu
1	Menu with no config options
2	No menu

These values are delcared as constants in the header files; `TWCONST.H` and `TWDECL.BAS`. Refer to *Appendix B - Constants on page 381* for a complete list of declared constants.

InSystem% specifies whether or not to add the option **Enable full menu** to the system menu. The settings are:

InSystem%	**Enable full menu**
0	Don't add this option.
1	Add

Note: *InSystem% is only valid when State% is either 1 or 2.*

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example 1 `SetMenuMode 0`

This is equivalent to the **View** menu option **Full menu**.

Example 2 `SetMenuMode 1`

This is equivalent to the **View** menu option **No config options**.

Example 3 `SetMenuMode 2, 1`

This is equivalent to the **View** menu command **No menu** with **Change via system menu** set in **Session Preferences**.

Example

Menu View > Full menu

View > No config options

View > No menu

See also **SetWindowStyle**, **ShowButtons**, **ShowMenu**, **ShowModuleLabels**, **ShowStatus**

See also **ttSetWindowStyle**, **ttShowButtons**, **ttShowMenu**, **ttShowModuleLabels**, **ttShowStatus**

ttSetStatusMsg ()

Description Sets the status message.

Declaration **Sub SetStatusMsg** (**ByVal** *Msg$* **As String**)

Visual Basic
ttSetStatusMsg
(**ByVal** *HSess* **As Long**,
ByVal *Msg* **As String**)
As Long

C language
int ttSetStatusMsg
(**HSess** *HSess*,
LPCSTR *Msg*);

Remarks This message is displayed in the Bottom Status bar only when there no system state messages to be displayed.

Parameter *Msg$* is the text string to be displayed in the Bottom Status bar.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
SetStatusMsg  "Testing script"
```

The message, “Testing script” is displayed in the Bottom Status bar, left justified.

Example

See also **GetStatusMsg**, **SetTitle**

See also **ttGetStatusMsg**, **ttSetTitle**

ttSetTitle ()

Description Sets the current TTWIN title.

Declaration **Sub SetTitle** (**ByVal** *Title$* **As String**)

Visual Basic

ttSetTitle
(**ByVal** *HSess* **As Long**,
ByVal *Title* **As String**)
As Long

C language

int ttSetTitle
(**HSess** *HSess*,
LPCSTR *Title*);

Parameter *Title$* is a text string assigned to the TTWIN window title. This text is added to the prefix "TTWIN - ", which always exists.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
SetTitle  "Oracle Host"
```

The TTWIN window will have the title, “TTWIN - Oracle Host”.

Example

Menu Configure > Title

See also **SetStatusMsg**

See also **ttSetStatusMsg**

ttSetWindowStyle ()

Description Changes the window style of the main TTWIN window.

Declaration **Sub SetWindowStyle** (**ByVal** *Style%* **As Integer**)

Visual Basic
ttSetWindowStyle
(**ByVal** *HSess* **As Long**,
ByVal *Style* **As Long**)
As Long

C language
int ttSetWindowStyle
(**HSess** *HSess*,
int *Style*);

Parameter The window *Style%* setting are as follows:

Style%	Window style
0	Hide
1	Normal (activate) show in the most recent size & position (after either minimise or maximise).
2	Minimise (activate).
3	Maximise (activate).
4	Normal (don't activate).
5	Show (activate). Opposite of Hide.
6	Minimise (don't activate).

Where:

Activate specifies that TTWIN becomes the active window.

Don't activate specifies that the currently active window remains active.

These values are delcared as constants in the header files; `TWCONST.H` and `TWDECL.BAS`. Refer to *Appendix B - Constants on page 381* for a complete list of declared constants.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
SetWindowStyle 2
```

The current TTWIN window sill be minimised.

Example

See also **SetMenuMode**, **ShowButtons**, **ShowMenu**, **ShowModuleLabels**, **ShowStatus**

See also **ttSetMenuMode**, **ttShowButtons**, **ttShowMenu**, **leLabels**, **ttShowStatus**

Sgn ()

Description Returns an integer that is either -1 or +1, indicating the sign of a number.

Declaration **Function Sgn** (**ByVal** *Num%* **As Integer**) **As Integer**

Return value The integer returned is:

Value	Sign
-1	*Num%* is negative.
+1	*Num%* is positive.

Example 1

```
Sgn (-5)
```

Returns -1.

Example 2

```
Sgn (123)
```

Returns 1.

Shell

Description Executes another application.

Declaration **Sub Shell** (**ByVal** *Path$* **As String**[, **ByVal** *WinStyle%* **As Integer**])

Parameters *Path$* is a string that specifies the executable file. The path name can include the directory and drive, and any arguments or command line switches.

The *WinStyle%* setting are as follows:

WinStyle%	Window style
0	Hide (activate).
1	Show (activate) in the most recent size & position (after either minimise or maximise).
2	Minimise (activate).
3	Maximise (activate).
4	Normal (don't activate).
6	Minimise (don't activate).

Where:

Activate specifies that TTWIN becomes the active window.

Don't activate specifies that is currently active window remains active.

Example

```
Shell "c:\excel\excel.exe", 4
CNo = DDEInitiate ("Excel", "Sheet1")
```

The above statements start Excel and initiate a DDE conversation.

ttShowButtons ()

Description Shows or hides any visible button bars in the TTWIN main window.

Declaration **Sub ShowButtons** (**ByVal** *Show%* **As Integer**)

Visual Basic **ttShowButtons**

(**ByVal** *HSess* **As Long**,

ByVal *Show* **As Long**)
As Long

C language

int ttShowButtons
(**HSess** *HSess*,
BOOL *Show*);

Parameter

The possible values for *Show%* are:

Show%	**Button set state**
0	Hide currently visible button set(s).
1	Show button set(s).

Return value

The function returns the following values:

Value	**Meaning**
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
ShowButtons  1
```

Show all button sets which have been set visible within the **Configure Buttons** option.

Menu

View > Buttons

See also

SetMenuMode, **SetWindowStyle**, **ShowMenu**, **ShowModuleLabels**, **ShowStatus**

See also

ttSetMenuMode, **ttSetWindowStyle**, **ttShowMenu**, **ttShowModuleLabels**, **ttShowStatus**

ttShowMenu ()

Description Shows or hides the menus in the TTWIN main window.

Declaration **Sub ShowMenu** (**ByVal** *Show%* **As Integer**)

Visual Basic

ttShowMenu
(**ByVal** *HSess* **As Long**,
ByVal *Show* **As Long**)
As Long

C language **int ttShowMenu**
(**HSess** *HSess*,
BOOL *Show*);

Parameter The possible values for *Show%* are:

Show%	Menu state
0	Hide the menu.
1	Show the menu.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example
```
ShowMenu 0
```

Example

See also **SetMenuMode**, **SetWindowStyle**, **ShowButtons**, **ShowModuleLabels**, **ShowStatus**

See also **ttSetMenuMode**, **ttSetWindowStyle**, **ttShowButtons**, **ttShowModuleLabels**, **ttShowStatus**

ttShowModuleLabels ()

Description Shows or hides the Top Status bar in the TTWIN main window.

Declaration **Sub ShowModuleLabels** (**ByVal** *Show%* **As Integer**)

Visual Basic

ttShowModuleLabels
(**ByVal** *HSess* **As Long**,
ByVal *Show* **As Long**)
As Long

C language

int ttShowModuleLabels
(**HSess** *HSess*,
BOOL *Show*);

Remarks The Top Status bar contains the currently selected emulation, communications and file transfer module names as well as the current state of the connection.

Parameter The possible values for *Show%* are:

Show%	Top Status bar state
0	Hide the Top Status bar.
1	Show the Top Status bar.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
ShowModuleLabels 0
```

Hides the Top Status bar.

Example

Menu View > Top status bar

See also **SetMenuMode**, **SetWindowStyle**, **ShowButtons**, **ShowMenu**, **ShowStatus**

See also **ttSetMenuMode**, **ttSetWindowStyle**, **ttShowButtons**, **ttShowMenu**, **ttShowStatus**

ttShowStatus ()

Description Shows or hides the Bottom Status bar in the TTWIN main window.

Declaration **Sub ShowStatus** (**ByVal** *Show%* **As Integer**)

Visual Basic

ttShowStatus
(**ByVal** *HSess* **As Long**,
ByVal *Show* **As Long**)
As Long

C language

int ttShowStatus
(**HSess** *HSess*,
BOOL *Show*);

Parameter The possible values for *Show%* are:

Show%	Bottom Status bar state
0	Hide the Bottom Status bar.
1	Show the Bottom Status bar.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
ShowStatus 0
```

Hides the Bottom Status bar.

Example

Menu View > Bottom status bar

See also **SetMenuMode**, **SetWindowStyle**, **ShowButtons**, **ShowMenu**, **ShowModuleLabels**

See also **ttSetMenuMode**, **ttSetWindowStyle**, **ttShowButtons**, **ttShowMenu**, **ttShowModuleLabels**

Sin ()

Description Calculates the sine of an angle.

Declaration **Function Sin** (**ByVal** *Angle* **As Single**) **As Single**
Function Sin (**ByVal** *Angle* **As Double**) **As Double**

Parameter *Angle* is the angle in radians.

Return value Returns the sine of *Angle*.

Example
```
x = Sin          '(Radians)
```

See also **Atn**, **Cos**, **Tan**

Space$ ()

Description Returns a string with the specified number of spaces.

Declaration **Function Space$** (**ByVal** *Spaces%* **As Integer**) **As String**

Function Space (**ByVal** *Spaces%* **As Integer**) **As Variant**

Parameter *Spaces%* is the number of space required.

Return value Returns a string of *Spaces%* spaces.

Example

```
BlankField = Space (12)
Data = Field1 & BlankField & Field2
```

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **StrComp**, **String$, Trim$**, **Val**

Sqr ()

Description Calculates the square root of a number.

Declaration **Function Sqr** (**ByVal** *Number* **As Single**) **As Single**
Function Sqr (**ByVal** *Number* **As Double**) **As Double**

Parameter *Number* is any valid numeric expression greater than or equal to zero.

Return value Returns the square root of *Number*.

Example
```
SquareRoot = Sqr (100)
```
SquareRoot contains 10.

ttStartCapture ()

Description Start capturing the data stream being sent from your remote host to TTWIN.

Declaration **Sub StartCapture** (**ByVal** *FileName$* **As String**[, **ByVal** *Flags%* **As Integer**])

Visual Basic
ttStartCapture
(**ByVal** *HSess* **As Long**,
ByVal *FileName* **As String**,
ByVal *Flags* **As Long**)
As Long

C language **int ttStartCapture**
(**HSess** *HSess*,
LPCSTR *FileName*,
unsigned *Flags*);

Parameter *FileName$* is ignored when capturing to the clipboard.

Flags% (optional) is a combination of:

Flags%	Capture mode
0 or not specified	If *Flag%* is or not specified the default values are used, which are: • Capture to a file. • Overwrite an existing file. • Capture all characters (raw mode capture).
1	Capture to the clipboard.
2	Append capture to the end of the file.
4	Capture only printable characters.

These values are delcared as constants in the header files; `TWCONST.H` and `TWDECL.BAS`. Refer to *Appendix B - Constants on page 381* for a complete list of declared constants.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
StartCapture  "C:\TTWIN\TTWIN.CAP"
```

Start a raw mode capture to a file called `TTWIN.CAP` in the `C:\TTWIN` directory. An existing file of the same name, in the same directory, is overwritten.

Example

See also **StopCapture**

See also **ttStopCapture**

ttStartDDEServer ()

Description Starts TTWIN servicing DDE requests.

Declaration **Sub StartDDEServer**

Visual Basic **ttStartDDEServer** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttStartDDEServer** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
StartDDEServer
```

Example

See also **SetDDEServerName**, **StopDDEServer**

See also **ttSetDDEServerName**, **ttStopDDEServer**

Stop

Description Suspends execution.

Declaration **Sub Stop**

Remarks **Stop** does not clear variables nor close any files unless the statement is in a compiled executable file.

Example

```
Stop
```

ttStopCapture ()

Description Stop capturing the data stream being sent from your remote host to TTWIN.

Declaration **Sub StopCapture**

Visual Basic **ttStopCapture** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttStopCapture** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
StopCapture
```

Example

Menu Action > Stop capture

See also **StartCapture**

See also **ttStartCapture**

ttStopDDEServer ()

Description Stops TTWIN servicing DDE requests.

Declaration **Sub StopDDEServer**

Visual Basic **ttStopDDEServer** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttStopDDEServer** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Example

```
StopDDEServer
```

Example

See also **SetDDEServerName**, **StartDDEServer**

See also **ttSetDDEServerName**, **ttStartDDEServer**

Str$ ()

Description Returns a string that is the character version of a numeric expression.

Declaration **Function Str$** (**ByVal** *NumExp* **As Single**) **As String**
Function Str$ (**ByVal** *NumExp* **As Double**) **As String**
Function Str (**ByVal** *NumExp* **As Single**) **As Variant**
Function Str (**ByVal** *NumExp* **As Double**) **As Variant**

Parameter *NumExp%* is any valid numeric expression.

Return value Returns a string that is the character version of *NumExp%*.

Example
```
StringVar = Str$ (-987654321)
```
StringVar contains the string "-987654321".

See also **Asc**, **Chr$**

StrComp ()

Description Returns the results of a string comparison.

Declaration **Function StrComp** (**ByVal** *StringOne$* **As String**, **ByVal** *StringTwo$* **As String**[, **ByVal** *Type%* **As Integer**]) **As Variant**

Parameters *StringOne$* and *StringTwo$* are the strings to be compared.

Type% is the type of comparison to be performed, which can be:

Type%	Comparison type
0	Binary
1	Textual (case-insensitive)

Return value The possible return values are:

Value	Result
-1	*StringOne$* < *StringTwo$*
0	*StringOne$* = *StringTwo$*
1	*StringOne$* > *StringTwo$*
Null	*StringOne$* or *StringTwo$* is **Null**.

Example

```
Name1 = "john"
Name2 = "JOHN"
Result1 = StrComp (Name1, Name2, 0)
Result2 = StrComp (Name1, Name2, 1)
```

Result1 contains 1. *Result2* contains 0.

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **String$, Trim$**, **Val**

String$ ()

Description Returns a string of a repeated character.

Declaration **Function String$** (**ByVal** *Len%* **As Integer**, **ByVal** *Asc%* **As Integer** | **ByVal** *Sample$* **As String**) **As String**

Function String (*Len%* **As Integer**, *Asc%* **As Integer** | *Sample$* **As String**) **As Variant**

Parameters *Len%* is the number of repetitions of the character.

Asc%, the ASCII value of a character. Alternatively, *Sample$* is a string the first character of which is used for the repetition string.

Return value Returns a string of *Len%* repetitions of the character *Asc%* or the first character in *Sample$*.

Example 1

```
String$ (10, 65)
```

Returns "AAAAAAAAAA".

Example 2

```
String$(10, "Catastrophe")
```

Returns "CCCCCCCCCC".

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **Trim$**, **Val**

Tan ()

Description Calculates the tan of an angle.

Declaration **Function Tan** (**ByVal** *Angle* **As Single**) **As Single**
Function Tan (**ByVal** *Angle* **As Double**) **As Double**

Parameter *Angle* is the angle in radians.

Return value Returns the tan of *Angle*.

Example
```
x = Tan '(Radians)
```

See also **Atn**, **Cos**, **Sin**

Text

Description Defines a line of text on a dialog box.

Syntax **Text** *x, y, cx, cy, Text$*

Parameters *x, y* are the horizontal (*x*) and vertical (*y*) position of top left corner of the text, relative to dialog box.

cx, cy are the width (*cx*) and height (*cy*) of the text.

Text$ is the text string to be placed in the dialog box.

Example `Text 14, 14, 41, 8, "User name:"`
The text “User name:” appears in the location specified.

See also **CheckBox**, **ComboBox**, **GroupBox**, **ListBox**, **TextBox**

TextBox

Description Defines a text box for user input.

Syntax **TextBox** [**NoEcho**] *x, y, cx, cy, Input$*

Remarks A text box is only used for input and input echo. To display text beside the text box use **Text**.

Parameters If **NoEcho** is specified, text appears as '*' characters as it is being typed.

x, y are the horizontal (*x*) and vertical (*y*) position of top left corner of the text box, relative to dialog box.

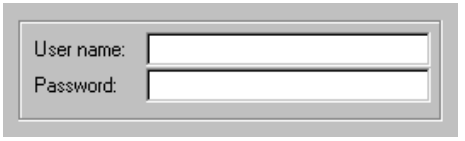
cx, cy are the width (*cx*) and height (*cy*) of the text box.

Input$ is the string variable that contains the text entered.

Example

```
GroupBox        8, 4, 166, 40,     ""
Text            14, 14, 41, 8,     "User name:"
Text            14, 27, 36, 8,     "Password:"
TextBox         58, 12, 111, 12,   UserName$
TextBox NoEcho 58, 25, 111, 12,    Password$
```

The dialog box will contain two text boxes. The first text box returns a user name string through the variable *UserName$*. The user name will be visible when typed. The other text box returns a string through *Password$* and the text will appear as * when typed.



See also **Begin Dialog**, **CheckBox**, **ComboBox**, **GroupBox**, **OptionButton**, **ListBox**, **Text**

Timer ()

Description Returns the time, in seconds, since midnight.

Declaration **Function Timer As Long**

Return value Returns the time, in seconds, since midnight.

Example

```
Elapse = Timer
```

Trim$ ()

Description Returns a string with all leading and trailing spaces, tabs and linefeeds removed.

Declaration **Function Trim$** (**ByVal** *String$* **As String**) **As String**
Function Trim$ (**ByVal** *String$* **As String**) **As Variant**

Parameter *String$* is any valid string.

Return value Returns *String$* with all the leading and trailing spaces, tabs and linefeeds removed.

Example
```
Trim$ ("   Hello there     ")
```
Returns “Hello there”.

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$**, **Val**

TTWinExit

Description Shutdowns the current TTWIN session.

Declaration **Sub TTWinExit**

Visual Basic **ttExit (ByVal** *HSess* **As Long) As Long**

C language **int ttExit (HSESS** *HSess***)**

Example
```
TTWinExit
```

Menu File > Exit

UBound ()

Description Returns the upper bound of the specified dimension for an array.

Declaration **Function UBound** (**ByVal** *Array* **As Any**[, **ByVal** *Dimension%* **As Integer**]) **As Integer**

Parameters *Array* is any valid array variable.

Dimension% indicates the dimension for which the upper bound is to be returned. Use 1 for the first dimension, 2 for the second, etc. If *Dimension%* is not specified, 1 is assumed.

Return value Returns the upper bound for *Array* and *Dimension%* (if specified).

Example

```
Dim Data (-1 to 1, 25 to 100)
Upper1 = UBound (Data, 1)
Upper2 = UBound (Data, 2)
```

Upper1 contains 1 and *Upper2* contains 100.

See also **LBound**

UCase$ ()

Description Returns a string with every lowercase character converted to uppercase.

Declaration **Function UCase$** (**ByVal** *String$* **As String**) **As String**
Function UCase (**ByVal** *String$* **As String**) **As Variant**

Parameter *String$* is the string to be converted into uppercase.

Return value Returns *String$* as an uppercase string.

Example
```
UCase ("Hello There")
```
Returns “HELLO THERE”.

See also **LCase$**

ttUnlockFlow ()

Description Unlocks the communication flow control.

Declaration **Sub UnlockFlow**

Visual Basic **ttUnlockFlow** (**ByVal** *HSess* **As Long**) **As Long**

C language **int ttUnlockFlow** (**HSess** *HSess*);

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Menu Action > Unlock flow control

Example

```
If Condition Then UnLockFlow
```

Example

See also

Val ()

Description Returns the number in a string.

Declaration **Function Val** (**ByVal** *String$* **As String**) **As Integer**

Remarks The dot (**.**) is recognise as a decimal separator and blanks, tabs and linefeeds are stripped from the string first. **Val** stops searching the string for a number as soon as it finds a character that is not recognised as part of a number.

Parameter *String$* is the string to be searched for a number.

Return value Returns the integer in *String$*.

Example 1 `Val ("-123")`
Returns -123.

Example 2 `Val (“574.3 multiplied by 21”)`
Returns 574.3

Example 3 `Val (“574 1973”)`
Returns 5741973

See also **InStr**, **Left$**, **Len**, **LTrim$**, **Mid$**, **Replace$, Right$**, **RTrim$**, **Space$**, **StrComp**, **String$**, **Val**

ttWaitForResponse () C Only

Description Wait for a response from the host or until a timeout occurs.

C language
int ttWaitForResponse
(**HSess** *HSess*,
LPCSTR **Strings*,
int *StringCount*,
UINT *Timeout*);

Parameters *Strings* is an array of pointers to strings to be matched to a string received from the host. Several text strings can be given.

StringCount is the number of strings.

Timeout is the maximum wait time for a string, in seconds. If *Timeout* is 0 then no timeout period is set and the wait is indefinite.

Return value The function returns the following values:

Value	Meaning
>=0	The function succeeded. An index of the matched string.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes including *timeout* (-14).

Example

See also

ttWaitForStrAsync C only

Description Waits for a string asynchronously.

C language

int ttWaitForStrASync C
HSESS *HSessS*
LPCSTR * *Strings*
int StringCount
int StringCount
FARPROC Event Handler
UINT *Timeout*

Parameter *Strings* is an array of pointers to strings to be matched to a string received from the host.

String count is the number of strings in the array.

Timeout is the maximum wait time in seconds. If *Timeout* is 0 then no *timeout* period is set and the wait is indefinite.

Event Handler is user defined handler which will be called to receive results of **ttWait ForStrAsync.** Event Handler must be declared as follows:

Void EfH (HSESS hs, Long Result);

hs identifies the session which was specified in the call **ttWait ForStrAsync**

Result - Result of wait function.

Value	Meaning
>=0	The function succeeded. An index of the string
< 0	The function failed including *timeout* (-14)

Return value The functions return the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed.)

Example

```
Void Far Pascal - export EvH (HSESS hS, Long Result)
     Char Msg [100]
     Wsprints (msg, "EvH received result of ttWait
     ForStrAsync.  Result=%ld", Result);
     Message Box (HWnd, Msg, "Final Result", MB-OK)
     Const char * Strings [3] = {"Login:", "123",
     "ABC"}; int i;
     i = ttWait ForStrAsync (HSESS, Strings, 3,
     (FARPROC) Ev H,0); if (i<0)
     something wrong
     else
     waits for string
```

ttWaitForStrCancel C Only

Description Cancels waiting for strings immediately. Especially important for **ttWaitForStrAsync** when *Timeou*t was set to 0.

C language **int ttWaitForStrCancel (***HSess HSess***)**;

Return value The functions return the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed.)

ttWaitForString ()

Description Waits for a string (from a set of strings) to be received from the host or until a timeout occurs.

Declaration **Function WaitForString** (**ByVal** *Timeout%* **As Integer**, **ByVal** *String$* **As String**[, **ByVal** *String$* **As String**...]) **As Integer**

Visual Basic

ttWaitForString
(**ByVal** *HSess* **As Long**,
ByVal *Data* **As String**,
ByVal *Timeout* **As Long**)
As Long

C language **int ttWaitForString**

(**HSess** *HSess*,
LPCSTR *Data*,
UINT *Timeout*);

Parameters

Timeout% is the maximum wait time for a string, in seconds. If *Timeout%* is 0 then no timeout period is set and the wait is indefinite.

String$ is a text string to be matched to a string received from the host. Several text strings can be given, each being separated by a comma.

Remarks

In Visual Basic or C Language, only 1 string can be used. In C Language use **ttWaitForResponse** to wait for one of the sets of strings.

Return value

On completion, an integer value is returned:

Value	Meaning
0	Timed out. No match was achieved within the specified *Timeout%* period.
1	First string matched
2	Second string matched
etc.	...

Return Value in Visual Basic and C Language:

Value	Meaning
>=0	The function succeeded.
<0	The function failed, including *timeout* (-14)

Example

```
x = WaitForString (10, "login:","host not found")
```

x will return the following:

x	Meaning
0	If after ten seconds no match is found.
1	If the string "login:" is detected.
2	If the string "host not found" is detected.

While

Description Repeats the execution of a group of statements while a condition remain true.

Syntax
While *Expression*
Statement(s)
Wend

Parameters Each time the **While** loop is executed, beginning with the first time it is entered, the *Expression* is evaluated. As in the **If** statement, if *Expression* is TRUE (non zero) then S*tatement(s)* is executed. If *Expression* is FALSE (zero), the next statement after the **Wend** statement is executed.

Example 1
```
While 1
     Display ("Hello")
Wend
```
Note: *This loop would print the word "Hello" forever.*

Example 2
```
Counter = 3
While Counter
     Display ("Hello")
```

```
        Counter = Counter - 1
Wend
```

This loop will print the word “Hello” 3 times.

ttWriteFieldString ()

Description Writes data into the current field.

Visual Basic

ttWriteFieldString
(**ByVal** *HSess* **As Long**,
ByVal *Value* **As String**,
ByVal *Length* **As Long**) **As Long**

C language

int ttWriteFieldString
(**HSess** *HSess*,
LPCSTR *Value*,
unsigned *Length*);

Parameter *Value* is the data to be written into the current field - if the data supplied is bigger than the field area, it is truncated.

Length is the length of the data to be written.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

Access to TTWin modules config parameters

Functions **ttGetTtwinCfgNames ttGetTtwinCfgStrings ttGetTtwinCfg-String ttSetTtwinCfgStrings ttSetTtwinCfgString**

ttGetEmulCfgNames ttGetEmulCfgStrings ttGetEmulCfg-String ttSetEmulCfgStrings ttSetEmulCfgString

ttGetComCfgNames ttGetComCfgStrings ttGetComCfg-String ttSetComCfgStrings ttSetComCfgString

ttGetFTCfgNames ttGetFTCfgStrings ttGetFTCfgString ttSetFTCfgStrings ttSetFTCfgString

Description This is a set of functions that can be used to work with modules config variables. In general all these functions are the same. The difference in the name points to the type of module which config variables will be accessed. Therefore below will be described 5 abstract functions:

Functions **ttGet**CfgNames**
TtGetCfgStrings**
TtGetCfgString**
TtSetCfgStrings**
TtSetCfgString**

Where "**" points to module . Also see the full list of functions above:

"Ttwin" for ttwin.exe
"Emul" for emulator modules
"Com" for communication modules
"FT" for file transfer modules

Return value Possible return error codes (<0):

Value	Meaning
TE_NOLINK	There is not access to the module variables.Most likely module does not support these functions.
TE_CFGBUFFTOOSMALL	Size of the buffer in module is too small to set this value.
TE_NULLPOINTERACCESS	Address of either module's config variable or Names = NULL
TE_NULLCFGACCESSTABLE	Module does not have CfgAccessTable, which is used to exchange values between Names and module variable.
TE_CFGPARAMNOTFOUND	Parameter name was not found in the CfgAccessTable.
TE_CFGPARAMNOVALUE	When we use one of ttSet…. function parameter does not have value, which must be set.
TE_CANNOTSETIFCONNECTED	Normally only for communication modules. We can not set values if it is in connected state
TE_FAIL	Any other errors

ttGet**CfgNames

Declaration **Function Get**CfgNames (***Names*** As String, ***Size*** As Integer) As Integer**

Visual Basic **Function ttGet**CfgNames (ByVal *HSess* As Long,**
ByVal *Names* As String,

ByVal *Size* **As Long) As Long**

C language **int ttGet**CfgNames (HSESS** *HSess*, **LPSTR** *Names*, **int** *Size* **)**

Parameter *Names* is a buffer to receive the list of config parameter names for the module.

Normally must be the same as the names for this module in `*.twc` file.

Names are separated by character \0 (0x00) and two this characters at the end of the list.

Example

```
LPSTR CfgNames ="Default-
Mode\0LanguageName\0\0"
```

Size is the size of the buffer Names.

Return value The function returns the following values:

Value	Meaning
>=0	the function succeeded. Actual number of bytes copied to buffer Names.
0<	the function failed. See above.

Example

```
Dim sCfgNames As String
Dim I As Integer
sCfgNames=String$(1024,32)
I = GetTtwinCfgNames(sCfgNames,1024)
```

ttGet**CfgStrings

Declaration **Function ttGet**CfgStrings (CfgStrings As String,** *Size* **As Integer) As Integer**

Visual Basic **Function ttGet**CfgStrings (ByVal** *HSess* **As Long,**
ByVal CfgStrings As String,
ByVal *Size* **As Long) As Long**

C language **int ttGet**CfgStrings (HSESS** *HSess*, **LPSTR CfgStrings, int** *Size* **);**

Parameter *CfgStrings* is a buffer to receive the list of config parameter strings for the module. Normally must be the same as the strings for this module in *.twc file.

Strings are separated by character \0 (0x00) and two this characters at the end of the list.

Example

```
LPSTR CfgStrings ="Default-
Mode=80\0LanguageName=English\0\0"
```

Size is the size of the buffer CfgStrings.

Return value The function returns the following values:

Value	Meaning
>=0	the function succeeded. Actual number of bytes copied to buffer CfgStrings
0<	the function failed. See above.

```
Dim sCfgStrings As String
Dim I As Integer
sCfgStrings=String$(2048,32)
I = GetTtwinCfgStrings (sCfgNames,2048)
```

ttGet**CfgString

Declaration **Function ttGet**CfgString (CfgString As String,** *Size* **As Integer) As Integer**

Visual Basic **Function ttGet**CfgString (ByVal** *HSess* **As Long,**
ByVal CfgString As String,
ByVal *Size* **As Long) As Long**

C language **int ttGet**CfgString (HSess** *HSess***, LPSTR CfgString, int Size);**

Parameter *CfgString* is a buffer to receive the requested config parameter string for the module.

Normally must be the same as the strings for this module in `*.twc` file.

CfgString contents config parameter name on call to the `ttGet**CfgString.`

When function returns CfgString contents config parameter string.

Example

```
on call CfgString  = "DefaultMode"
 Returns CfgString = "DefaultMode=80\0"
```

Size is the size of the buffer CfgStrings.

Return value The function returns the following values:

Value	Meaning

>=0	the function succeeded. Actual number of bytes copied to buffer CfgStrings.
0<	the function failed. See above.

Example

```
Dim sCfgString As String
Dim I As Integer
sCfgString=String$(200,32)
sCfgString = "DefaultMode"
I = GetTtwinCfgString (sCfgString, 200)
```

ttSet**CfgStrings

Declaration **Function ttSet**CfgStrings (CfgString As Strings,** *Size* **As Integer) As Integer**
Function TtSet CfgStrings (ByVal** *HSess* **As Long, ByVal CfgStrings As String, ByVal** *Size* **As Long) As Long**

C language **int TtSet**CfgStrings (HSESS** *HSess***, LPSTR CfgStrings, int** *Size* **);**

Parameter *CfgStrings* is a buffer with the list of config parameter strings to set config parameter values of the module.

Normally must be the same as the strings for this module in `*.twc` file.

Example

```
CfgStrings ="Default-
Mode=80\0LanguageName=English\0\0"
```

Size is the actual size of config parameter strings.

Return value The function returns the following values:

Value	Meaning
>=0	the function succeeded.
0<	the function failed. See above.

Example

```
Dim sCfgStrings As String
Dim I As Integer
sCfgStrings = "Default-
Mode=80\0LanguageName=English\0\0"
I = SetTtwinCfgStrings (sCfgStrings, 37)
```

ttSet**CfgString

Declaration **Function ttSet**CfgString (CfgString As String,** *Size* **As Integer) As Integer**

Visual Basic **Function TtSet**CfgString (ByVal** *HSess* **As Long,**
ByVal CfgString As String,
ByVal *Size* **As Long) As Long**

C language **int TtSet**CfgString (HSESS** *HSess*, **LPSTR CfgString, int Size);**

Parameter *CfgString* is a buffer with the config parameter string to set config parameter value of the module.

Normally must be the same as the string for this module in `*.twc` file.

Example

```
CfgString ="DefaultMode=80\0\0"
```

Size is the actual size of config parameter string.

Return value The function returns the following values:

Value	Meaning
>=0	the function succeeded. Actual number of bytes copied to buffer CfgStrings.
0<	the function failed. See above.

Example

```
Dim sCfgString As String
Dim I As Integer
sCfgString = "DefaultMode=80\0\0"
I = SetTtwinCfgString (sCfgString, 16)
```

Switch module functions

Functions **ttSwitchEmulator ttSwitchComm ttSwitchFTransfer**

Description This set of functions is used to switch ttwin module. Using of all these functions are the same. The deference in the name points to the type of module which will be switched. Therefore below will be described only one abstract **ttSwitch**** function,
where part of the name "**" can be :

"Emulator"
"Comm"
"FTransfer"

See the full list of functions above.

Visual Basic **Function ttSwitch** (ByVal** *HSess* **As Long,**
ByVal *ModuleName* **As String) As Long**

C language **int ttSwitch** (HSESS** *HSess*, **LPSTR** *ModuleName***);**

Parameter *ModuleName* is string which contents the name of the module with must be loaded. Normally it is the file name of the module without extension.

Return value The function returns the following values:

Value	Meaning
>=0	the function succeeded. Actual number of bytes copied to buffer CfgStrings.
0<	the function failed. See the full list of return values in the appendix ???.

Example

```
Dim ModuleName As String
Dim RetVal As Long
ModuleName="VT220"
RetVal = ttSwitchEmulator(HSess, ModuleName)
```

Get**CfgNames

Description

Declaration

Visual Basic

ttWriteFieldString
(**ByVal** *HSess* **As Long**,
ByVal *Value* **As String**,
ByVal *Length* **As Long**) **As Long**

C language

int ttWriteFieldString
(**HSess** *HSess*,
LPCSTR *Value*,

unsigned *Length*);

Parameter *Value* is value of the field currently selected.?????????????

Length is the number of characters in Value.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Refer to *Appendix B - Constants on page 381* for a complete list of return codes.

AttachTtwin

Description Attaches the existing ttwin session to the ttwin activeX control. This method is used to switch ttwin session between activeX controls.

Visual Basic

```
AttachTtwin (Kbd As Long,
             Scrn As Long,
             Attach As Long) As Integer
```

Parameter *Kbd, Scrn, Attach* are internal Ttwin ActiveX handles, which are used to assosiate ttwin (ttwin.exe) session and Ttwin ActiveX control. Values of these handles must be get from other Ttwin ActiveX control, which has an attached ttwin session.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded. TTWin session was attached to the control.

Value	Meaning
< 0	The function failed. Control already has an attached TTWin session.

Example This example shows how to switch ttwin session between TTWin ActiveX controls from Ttwin1to the Ttwin2.

```
Dim Kbd As Long
Dim Scrn As Long
Dim Att   As Long
Dim I As Integer

I = Ttwin1.GetAttachInfo(Kbd, Scrn, Att)
I = Ttwin1.DeAttachTtwin
I = Ttwin2.AttachTtwin(Kbd, Scrn, Att)
…
```

See also **GetAttachInfo, DeAttachTtwin**

GetAttachInfo

Description This method is used to get internal Ttwin ActiveX handles which are used to attach ttwin session to the ttwin activeX control.

This method is used to switch ttwin session between activeX controls.

Visual Basic

```
GetAttachInfo (lpKbd As Long,
               lpScrn As Long,
               lpAttach As Long) As Integer
```

Parameter *lpKbd, lpScrn, lpAttach* are variables to get internal Ttwin ActiveX handles, which are used to assosiate ttwin (ttwin.exe) session and

Ttwin ActiveX control. After this method returns this variables can be used to attach ttwin session.

Return value The function returns the following values:

Value	Meaning
0	The function succeeded.
< 0	The function failed. Control does not have an attached TTWin session.

Example This example shows how to switch ttwin session between Ttwin ActiveX controls from Ttwin1 to the Ttwin2.

```
…
Dim Kbd As Long
Dim Scrn As Long
Dim Att   As Long
Dim I As Integer

I = Ttwin1.GetAttachInfo(Kbd, Scrn, Att)
I = Ttwin1.DeAttachTtwin
I = Ttwin2.AttachTtwin(Kbd, Scrn, Att)
…
```

See also **AttachTtwin, DeAttachTtwin**

DeAttachTtwin

Description This method is used to deattach ttwin session from Ttwin ActiveX control. This method is used to switch ttwin session between activeX controls.

Visual Basic DeAttachTtwin () As Integer

Parameter

Return value The function returns the following values:

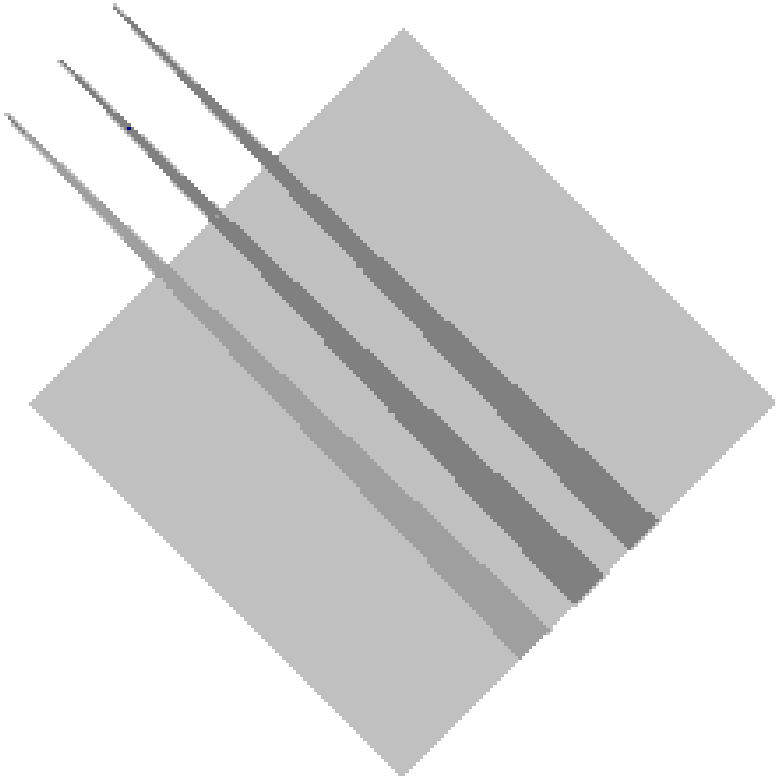
Value	Meaning
0	The function succeeded.
< 0	The function failed. Control does not have an attached TTWin session.

Example This example shows how to switch TTWin session between TTWin ActiveX controls from TTWin1
to the TTWin2.

```
…
Dim Kbd As Long
Dim Scrn As Long
Dim Att   As Long
Dim I As Integer

I = Ttwin1.GetAttachInfo(Kbd, Scrn, Att)
I = Ttwin1.DeAttachTtwin
I = Ttwin2.AttachTtwin(Kbd, Scrn, Att)…
```

See also **GetAttachInfo, AttachTtwin**

PART THREE
APPENDICES



Appendix A
Data Type Definitions

This appendix provides information on the storage size and range of the supported data types. For more information on data types and variable declarations refer to *Chapter 2 - Variables and Expressions on page 13.*

Integer

Integer variables hold numeric data in a two byte data format. Integers have a range of -32,768 to +32,767 inclusive.

Long (Long Integer)

Long integer variables hold numeric data in a four byte format. They have a range of -2,147,483,684 to +,147,483,647.

Single (Single precision floating point)

Single variables hold numeric data in a four byte format, with a range as follows:

Negative values: -3.402823^{38} to -1.401298^{-45}

Positive values: 1.401298^{-45} to 3.402823^{38}

Double (Double precision floating point)

Double variables hold numeric data in a four byte format with a range as follows:

Negative values: -1.79769313486232^{308} to -4.94065645841247^{-324}

Positive values: 4.94065645841247^{-324} to 1.79769313486232^{308}

String

String variables hold character data in a one byte per character format. The length is not predefined at the declaration of the variable. (Fixed length strings are not supported.)

Variant

Variants are a useful programming tool as they store data without regard to the format of the information. A **Variant** can contain characters and/or numbers. It is possible to predefine the type of a variant and TTWIN BASIC supports the following **Variant** data types:

Data type	Description
Empty	The value of a **Variant** before it has had a value or a string assigned to it. **Zero**, **Null** (see below) and **""** are not empty values. You can assign a variant back to **Empty** by assigning another **Empty** variant to it. The **Empty** value of a variant is usually used as a test condition in a routine.
Double	As described above.
Integer	As described above.
Long	As described above.
Null	A special data type used to identify a database field that has no content. If there is no data in the field that is assigned to a named variant then the variant will respond **Yes** to a **Null** interrogation.
Single	As described above.

Data type	Description
String	As described above.

Appendix B Constants

This appendix contains a list of the possible return codes for function that form the TTWIN DLL Interface. It also include a list of the ants and flags that are declared in the header files.

Return Codes	Value	Description
TE_OK	0	Successful operation
TE_FAIL	-1	Operation failed
TE_NOLINK	-2	No link to a TTWin session
TE_INVALIDSESSION	-3	Invalid session handle
TE_BUFFTOOSMALL	-4	Destination buffer is too small
TE_ALREADYDIALLING	-5	Cannot dial/hangup, already dialling
TE_NOMODEM	-6	Cannot dial/hangup, no modem selected
TE_ENTRYNOTFOUND	-7	Dialling/modem entry not found
TE_DIALTIMEOUT	-8	Dial operation timed out
TE_CANCELLED	-9	User cancelled the operation
TE_OUTPUTQUEUED	-10	Data was queued on a send/print
TE_NOTCONNECTED	-11	Attempted to send, but disconnected
TE_QUEUEOVERFLOW	-12	Send queue overflowed
TE_ALREADYWAITING	-13	Wait for strings when already waiting

Return Codes	Value	Description
TE_WAITTIMEOUT	-14	Wait for strings timed out
TE_ALREADYPRINTING	-15	Print request whilst already printing
TE_NOTHINGMARKED	-16	No marked region on copy/print marked
TE_UNKNOWNBSET	-17	No button set of that name
TE_ALREADYCAPTURING	-18	Capture already open
TE_UNKNOWNVAR	-19	No variable of that name

Other Flags	Value	For use with
TT_MENUFULL	0	**ttSetMenuMode**
TT_MENUNOCONFIG	1	**ttSetMenuMode**
TT_MENUNONE	2	**ttSetMenuMode**
TT_PRINTDFLT	0	**ttPrintClose**, **ttPrintScreen**, **ttPrintSelected**
TT_PRINTRAW	1	**ttPrintClose**, **ttPrintScreen**, **ttPrintSelected**
TT_PRINTWINDOWS	2	**ttPrintClose**, **ttPrintScreen**, **ttPrintSelected**
TT_CAPFILE	0	**ttStartCapture**
TT_CAPCLIPBOARD	1	**ttStartCapture**
TT_CAPOWRITE	0	**ttStartCapture**
TT_CAPAPPEND	2	**ttStartCapture**
TT_CAPRAW	0	**ttStartCapture**
TT_CAPVISUAL	4	**ttStartCapture**
TT_CURSSTEADY	80 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSBLINK	0 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**

Other Flags	Value	For use with
TT_CURSOFF	40 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSON	0 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSFULLBLOCK	1(Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSSMALLBLOCK	2 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSHORIZLINE	3 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_CURSVERTLINE	4 (Hex)	**ttGetCursorStyle**, **ttSetCursorStyle**
TT_WINSHOW	5	**ttSetWindowStyle**
TT_WINHIDE	0	**ttSetWindowStyle**
TT_WINMAXIMISE	3	**ttSetWindowStyle**
TT_WINMAXIMIZE	3	**ttSetWindowStyle**
TT_WINMINIMISE	2	**ttSetWindowStyle**
TT_WINMINIMIZE	2	**ttSetWindowStyle**
TT_WINRESTORE	9	
TT_BTNTOP	0	**ttBtnShowSet**
TT_BTNBOTTOM	1	**ttBtnShowSet**
TT_BTNLEFT	2	**ttBtnShowSet**
TT_BTNRIGHT	3	**ttBtnShowSet**
TT_BTNHORIZFLOAT	4	**ttBtnShowSet**
TT_BTNVERTFLOAT	5	**ttBtnShowSet**
TT_EMMODULE	1	
TT_CMMODULE	2	
TT_FTMODULE	3	
TT_STRINGVAR	1	
TT_INTVAR	2	

Appendix C
Reserved Words

TTWIN BASIC supports a larger number of reserved words. These consists of keywords, functions, operators, statements and any other words used by the **TTWIN** BASIC interpreter.

Note: *TTWIN BASIC is case sensitive.*

' (apostrophe)
Abs
Alias
And
Any
AppActivate
Append
Array
As
Asc
Assert
Atn
BasicLib
Begin
Begin Dialog
Beep
Bell
Break
BtnHideSet
BtnShowSet
Button
ButtonGroup
ByVal
Call
CancelButton
Caption
Case
CDbl
CfgNew
CfgOpen
CfgSave
CfgSaveAs
ChDir
ChDrive
CheckBox
Chr$
CInt
ClipboardCopy
ClipboardPaste
CLng
Close
CloseHostStream
ComboBox
Connect
Cos
CreatDDEItem
CSng
CStr
CVar
Declare
Dialog

DDEAdvise
DDEExecute
DDEInitiate
DDEPoke
DDERequest$
DDETerminate
DDETerminateAll
DDEUnAdvise
DestroyDDEItem
DialEntry
DialManual
Dialog
Dim
Disconnect
DispClearAll
DispClearSelect
DispHeight
Display
DispReadAttr
DispReadText
DispSelectAll
DispSelectCont
DispSelectRow
DispWidth
Do
Double
Else
ElseIf
EmKeys
EmReset
End
End Dialog
End If
End Function
End Sub
Environ$
Eof
Erl
Err
Error
Error$
Event
Exit
Exp
Explicit
FileAttr
FileGet
FileReceive
FileSend
Fix
For
FreeFile
Function
GetArgs
GetCursorCol
GetCursorRow
GetCursorStyle
GetDDEItem$
GetIniFile$
GetSessionCount
GetSessionFile$
GetSessionList
GetStatusMsg$
GetTitle$
GetTTWinPath$
GoSub
GoTo
GroupBox
Hangup
Hex$
If
Input
Input$
InStr
Int
Integer
Is
IsConnected
IsDDEItemDefined
IsEmpty
IsNull
IsNumeric
KeyboardLocked
Kill
LBound
LCase$
Left$
Len
Let
Lib
Line
ListBox
Lof
Log
Long
Loop

LTrim$
Mid$
MilliPause
MkDir
Mod
MsgBox
Name
Next
NoEcho
Not
Null
Oct$
OKButton
On
On Error
Open
OpenHostStream
Option
OptionButton
OptionGroup
Option Explicit
Or
Output
Pause
PhysicalKey
Print
PrintClose
PrintScreen
PrintSelected
Randomize
ReadHost$
ReadHostLine$
ReDim
Rem
Replace$
Resume
Reset
Return
Right$
RmDir
Rnd
RTrim$
Seek
Select
SelectDefaultSession
SelectHost
SelectModem
SelectSessionByID
SelectSessionByTitle
SendData
SendKeys
SetCursorStyle
SetDDEItem
SetDDEServerName
SetMenuMode
SetStatusMsg
SetTitle
SetWindowStyle
Sgn
Shared
Shell
ShowButtons
ShowMenu
ShowMenuMode
ShowModuleLabels
ShowStatus
Sin
Single
Space$
Sqr
Static
Step
StartCapture
StartDDEServer
Stop
StopCapture
StopDDEServer
Str$
StrComp
String
String$
Sub
Tan
Text
TextBox
Then
Timer
To
Trim$
TTWinExit
Type
UBound
UCase$
UnlockFlow
Until
Val

Variant
VirtualKey
Wait
WaitForString
Wend
While
Write

Appendix D
Summary

This appendix lists all the keywords, functions, statements and subroutines that make-up TTWIN. These elements have been grouped together based on their function. For a complete description of these items, refer to *Part Two - Language Reference.*

† - *indicates that the keyword, function, statement or subroutine is an implementation of Visual Basic.*

Button Sets

BtnHideSet **BtnShowSet**

Clipboard

ClipboardCopy **ClipboardPaste**

Communications

Break **Disconnect**
Connect **IsConnected**

Configuration Files

CfgNew **CfgOpen** **CfgSave**

CfgSaveAs

Conversion

†**Asc**
†**CDbl**
†**Chr$**
†**CInt**
†**CLng**
†**CSng**
†**CStr**
†**CVar**
†**Fix**
†**Hex$**
†**Int**
†**LCase$**
†**Oct$**
†**Str$**
†**UCase$**

Data Stream

CloseHostStream
Display
OpenHostStream
ReadHost
ReadHost$
ReadHostLine$
SendData
StartCapture
StopCapture
UnlockFlow
Wait
WaitForString

Data Types, Variables, Declarations and Expressions

†**Alias**
†**And**
†**Any**
†**Array**
†**As**
†**BasicLib**
†**ByVal**
†**Declare**
†**Dim**
†**Double**
†**Integer**
†**Is**
†**IsEmpty**
†**IsNull**
†**IsNumeric**
†**LBound**
†**Let**
†**Lib**
†**Long**
†**Mod**
†**Not**
†**Null**
†**Option Explicit**
†**Or**
†**ReDim**
†**Single**
†**Static**
†**String**
†**To**
†**Type**
†**UBound**
†**Variant**

DDE Client/Server Support

CreateDDEItem
DDEAdvise
DDEExecute
DDEInitiate
DDEPoke
DDERequest$
DDETerminate
DDETerminateAll
DDEUnAdvise
DestroyDDEItem
GetDDEItem$
IsDDEItemDefined
SetDDEItem
SetDDEServerName
StartDDEServer
StopDDEServer

Dialog Box

Begin Dialog...End Dialog
Button
ButtonGroup
CancelButton
Caption
CheckBox
ComboBox
Dialog
GroupBox
ListBox
†**MsgBox**
OKButton
OptionButton
OptionGroup
Text
TextBox

Directories, Files, Input and Output

†**Append**
†**ChDir**
†**ChDrive**
†**Close**
†**EOF**
†**FileAttr**
†**FreeFile**
†**Input**
†**Input$**
†**Kill**
†**Line Input**
†**LOF**
†**MkDir**
†**Name**
†**Open**
†**Output**
†**Print**
†**Reset**
†**RmDir**
†**Seek**
†**Shared**
†**Write**

Emulation

DispClearAll
DispClearSelect
DispHeight
DispSelectAll
DispSelectCont
DispSelectRect
DispReadAttr
DispReadText
DispWidth
EmKeys
EmReset
GetCursorCol
GetCursorRow
GetCursorStyle
KeyboardLocked
PhysicalKey
SetCursorStyle
VirtualKey

Error Handling

Assert
Erl
†**Err**
†**Error**
†**Error$**

File Transfers

FileGet
FileReceive
FileSend

Flow Control

Begin
†**Call**
†**Do...Loop**
†**End**
†**Exit**
†**For...Next...Step**
†**Function...End Function**
†**GoSub...Return**
†**GoTo**
†**If...Then...Else**
†**If...Then...ElseIf**
†**Let**
†**On Error**
†**Resume**
†**Select Case**
†**Stop**
†**Sub...End Sub**
†**Until**
†**While**
†**While...Wend**

Maths

†**Abs**
†**Atn**
†**Cos**
†**Exp**
†**Log**
†**Randomize**
†**Rnd**
†**Sgn**
†**Sin**
†**Sqr**
†**Tan**

Miscellaneous

†**AppActivate**
†**Bell**
†**Beep**
†**Environ$**
MillliPause
Pause
†**Rem**
†**SendKeys**
†**Shell**
Timer

Modem

DialEntry
DialManual
Hangup
SelectModem

Printing

PrintClose
PrintScreen
PrintSelected

String Manipulation

†**InStr**
†**Left$**
†**Len**
†**LTrim$**
†**Mid$**
Replace$
†**Right$**
†**RTrim$**
†**Space$**
†**StrComp**
†**String$**
†**Trim$**
†**Val**

Session

GetArgs
GetIniFile$
GetSessionCount
GetSessionFile$
GetList
GetStatusMsg$
GetTitle$
GetTTWinPath$
SelectDefaultSession
SelectHost
SelectSessionByID
SelectSessionByTitle
SetStatusMsg
SetTitle$
TTWinExit

Window controls

ShowButtons
ShowMenu
ShowModuleLabels
ShowStatus
SetMenuMode
SetWindowStyle

Appendix E
Customer Support

If you encounter any difficulty using TTWIN, or it does not operate as described, you should take the following steps:

1. Consult the manual and on-line help to make sure you are operating the program properly.

   Does your system meet the minimum system requirements?

   Check to see if the program is properly configured and the options for terminal emulation are correct.

2. Consult *Appendix G - Your Questions Answered on page 533* of the *TTWIN User's Guide.* This appendix includes answers to many of the most frequently asked questions. Perhaps the answers to your questions have been asked before.

3. Consult the reseller from whom you purchased TTWIN. Many resellers have trained technicians who can answer your questions.

4. If the answer you need is not available in the manual or from your reseller, email or fax a complete description of your problem to Turbosoft, marked to the attention of Technical Support.

Please include the TTWIN serial number, your name and phone number in case additional information is required.

fax: + 61 2 9319 0188
800 273 4995 (toll free US)
0800 898 169 (free call UK)

email: support@ttwin.com

5. You may also call Turbosoft Customer Services directly (24 hour service).

Phone: +61 2 9319 3988
800 272 9501 (toll free US)
0800 898 937 (free call UK)

Before calling, **make** sure you have the following at hand:

- Your software's serial number.
- Version numbers for:
  - TTWIN
  - Comms module
  - Emulation module

  (The version numbers can be obtained under the **Help About** drop down menu.)
- A computer running TTWIN.
- Be prepared to answer questions similar to the following:

Have you had TTWIN running successfully before? If so, have you changed any of your hardware or software since then?

Can you reproduce the problem?

Were any error messages displayed at the time the problem appeared? If so, what did it say?

Symbols

A

B

C

D

T

U

V

Variables
  declaring 14, 180, 290
  declaring arrays 16
  declaring strings 17
  dialog box 47
  dynamic arrays 16
  empty 253
  environment 200
  explicit 15, 280
  implicit 15
  null 255
  procedures relating to 390
  scope 17
  static 17
Variant 13, 378

W

Wait
  for string 358
Wend statement 27, 360
While statement 27, 360
Window
  procedures relating to 394
  setting the style 326