



NS Basic/CE Handbook

February 12, 2010

© NS BASIC Corporation, 2010.
71 Hill Crescent
Toronto, Canada M1M 1J3
+1 (416) 264-5999

This manual and the software described in it are copyrighted, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of NS BASIC Corporation. Under the law, copying includes translating into another language or format.

Every effort has been made to ensure that the information in this manual is accurate. NS BASIC Corporation is not responsible for printing or clerical errors. Specifications are subject to change without notice.

Microsoft, MS, Windows, and Windows CE are registered trademarks of Microsoft Corporation, registered in the United States and other countries.

Mention of third party products and their trademarks is for informational purposes only and constitutes neither an endorsement or recommendation. NS BASIC Corporation assumes no responsibility with regard to the performance or use of NS BASIC or these products.

Canadian Cataloguing In Publication Data

Darden, Marcus M., 1970-
NS BASIC/CE Handbook

Includes index.
ISBN 0-9695844-4-X

BASIC (Computer program Language).
2. Windows (Computer file) - Programming.
I. Henne, George W.P. 1954- . II. Title

QA76.73.B3D37 1998 005.4'3 C98-932304-8

LICENSE AGREEMENT

PLEASE READ THIS LICENSE CAREFULLY BEFORE USING THE SOFTWARE. BY USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, PROMPTLY RETURN THE PRODUCT TO THE PLACE WHERE YOU OBTAINED IT AND YOUR MONEY WILL BE REFUNDED.

1. License. The application, demonstration, system, and other software accompanying this License, whether on disk, in read only memory, or on any other media (the "Software"), the related documentation and fonts are licensed to you by NS BASIC Corporation ("NSBC"). You own the media on which the Software and fonts are recorded but NSBC and or NSBC's Licensor(s) retain title to the Software, related documentation and fonts. This License allows you to use the Software and fonts on a single Windows CE Product (which, for purposes of this License, shall mean a product bearing Microsoft's Windows CE logo), and make one copy of the Software and fonts in machine-readable form for backup purposes only. You must reproduce on such copy the NSBC copyright notice and any other proprietary legends that were on the original copy of the Software and fonts. You may also transfer all your license rights in the Software and fonts, the backup copy of the Software and fonts, the related documentation and a copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License.
2. Restrictions. The Software contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompile, reverse engineer, disassemble or otherwise reduce the Software to a human-perceivable form. You may not modify, network, rent, lease, load, distribute or create derivative works based upon the Software in whole or in part. You may not electronically transmit the Software from one device to another or over a network.
3. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Software and related documentation and fonts. This License will terminate immediately without notice from NSBC if you fail to comply with any provision of this License. Upon termination you must destroy the Software, related documentation and fonts.
4. Export Law Assurances. You agree and certify that neither the Software nor any other technical data received from NSBC, nor the direct product thereof, will be exported outside the United States except as authorized and as permitted by the laws and regulations of the United States. If the Software has been rightfully obtained by you outside of the United States, you agree that you will not reexport the Software nor any other technical data received from NSBC, nor the direct product thereof, except as permitted by the laws and regulations of the United States and the laws and regulations of the jurisdiction in which you obtained the Software.
5. Government End Users. If you are acquiring the Software and fonts on behalf of any unit or agency of the United States Government, the following provisions apply. The Government agrees: (i) if the Software and fonts are supplied to the Department

of Defense (DoD), the Software and fonts are classified as "Commercial Computer Software" and the Government is acquiring only "restricted rights" in the Software, its documentation and fonts as that term is defined in Clause 252.227-7013(c)(1) of the DFARS; and (ii) if the Software and fonts are supplied to any unit or agency of the United States Government other than DoD, the Governments' rights in the Software, its documentation and fonts will be as defined in Clause 52.227-19(c)(2) of the FAR or, in the case of NASA, in Clause 18-52.227-86(d) of the NASA supplement to the FAR.

6. NS BASIC will replace at no charge defective disks or manuals within 90 days of the date of purchase. NS BASIC warrants that the programs will perform generally in compliance with the included documentation. NS BASIC does not warrant that the programs and manuals are free from all bugs, errors or omissions.
7. Disclaimer of Warranty on Software. You expressly acknowledge and agree that use of the Software and fonts is at your sole risk. The Software, related documentation and fonts are provided "AS IS" and without warranty of any kind and NSBC and NSBC's Licensor(s) (for the purposes of provisions 7 and 8, NSBC and NSBC's Licensor(s) shall be collectively referred to as "NSBC") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. NSBC DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE SOFTWARE AND THE FONTS WILL BE CORRECTED. FURTHERMORE, NSBC DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE AND FONTS OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY NSBC OR A NSBC AUTHORIZED REPRESENTATIVE SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE SOFTWARE PROVE DEFECTIVE, YOU (AND NOT NSBC OR AN NSBC AUTHORIZED REPRESENTATIVE) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.
8. Limitation of Liability. Because software is inherently complex and may not be free from errors, you are advised to verify the work produced by the Program. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL NSBC BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE SOFTWARE OR RELATED DOCUMENTATION, EVEN IF NSBC OR A NSBC AUTHORIZED REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. In no event shall NSBC's total liability to you for all damages, losses, and causes of action

(whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Software and fonts.

9. Allocation of Risk: You acknowledge and agree that this Agreement allocates risk between you and NSBC as authorized by the Uniform Commercial Code and other applicable law and that the pricing of NSBC's products reflects this allocation of risk and the limitations of liability contained in this Agreement. If any remedy hereunder is determined to have failed of its essential purpose, all limitations of liability and exclusions of damages as set forth in this Agreement will remain in effect.
10. Support. NSBC may, at its option, provide support services at its standard fees for such services. Such support services will be governed by the limitations of liability under this Agreement.
11. Additional Restrictions: Any upgrade or enhancement of the program subsequently supplied by NSBC may only be used upon the destruction of the prior version, and shall be governed by the terms of this Agreement.
12. Controlling Law and Severability. This License shall be governed by and construed in accordance with the laws of the United States and the State of Delaware, as applied to agreements entered into and to be performed entirely within Delaware between Delaware residents. If for any reason a court of competent jurisdiction finds any provision of this License, or portion thereof, to be unenforceable, that provision of the License shall be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this License shall continue in full force and effect.
13. Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Software, related documentation and fonts, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by a duly authorized representative of NSBC.

CONTENTS

1. Introduction	9
1.1 What is BASIC?	9
1.1.1 NS Basic/CE	10
1.1.2 NS Basic/CE and Windows CE	10
1.2 System Requirements	10
1.3 Installation	11
1.3.1 Entering Your Serial Number	11
2. NS Basic/CE Concepts	13
2.1 Conventions Used in this Handbook	13
2.2 The Elements of an NS Basic/CE Program	14
2.2.1 Multi-Line Statements	14
2.2.2 Multiple Statements per Line	14
2.2.3 Literals, Data Types and Variables	14
2.2.4 Numeric Data Types	15
2.2.5 Boolean Data Type	15
2.2.6 Color Data Type	15
2.2.7 String Data Type	16
2.2.8 Array Data Type	16
2.2.9 Variable Names	16
2.3 Expressions and Operators	16
2.3.1 Arithmetic Operators	17
2.3.2 Relational Operators	18
2.3.3 Boolean Operators	18
2.4 FUNCTION and SUB Procedures	19
2.5 Projects, Modules, Forms, Objects and Controls	19
Controls and Objects	19
Object Events	20
Forms	20
Form Methods	21
Form Events	21
Program Properties and Events	22
Program Events	22
Modules	22
Hi Res Aware	22
Icons	23
3. Programming On the Device	25
3.1 Overview	25
3.2 Program Editor	25
3.3 Output Window	27
3.4 Visual Designer	27
3.4.1 Property Editor	30
3.4.2 Menu Editor	30
3.4.3 Notes	30
3.5 The NS Basic/CE Programming Environment	31
3.5.1 Creating a Program	31
3.5.2 Editing a Program	32

3.5.3 Formatting a program.....	32
3.5.4 Finding and Replacing	32
3.5.5 Overview	33
3.5.6 Running a Program	34
3.5.7 Debugging a Program.....	34
3.5.8 BREAK,Execute Function, Trace and Step.....	37
3.5.9 Saving and Loading a Program	38
3.5.10 Saving and Loading Programs as Text Files	38
4. Programming on the Desktop	39
4.1 Menu Options.....	39
4.2 Code Window	42
4.3 CE Screen	43
4.4 Project Explorer.....	44
4.5 Properties Window	45
4.6 Toolbox	46
4.7 Toolbar	47
4.8 Menu Editor.....	48
4.9 Options	50
4.9.1 Options – General	50
4.9.2 Options – Editor	51
4.9.3 Options – CE Screen	52
4.9.4 Options – Start	53
4.10 ActiveX Control Manager	54
4.11 Compiling from the Command Line.....	54
5. NS Basic/CE Reference	55
6. Advanced Topics	233
6.1 Coding Conventions.....	233
6.1.1 Naming guidelines.....	233
6.1.2 Text formatting guidelines.....	233
6.1.3 Comment guidelines	234
6.2 Handling Errors	234
6.2.1 Defensive programming.....	234
6.2.2 Error trapping	235
6.2.3 Error trapping file operations.....	235
A. Error Codes	237
B. Constants	239

1. Introduction

Welcome to NS Basic/CE for Windows CE. NS Basic/CE is designed to meet the needs of Windows CE users. It is a simple yet powerful language that can be used to write programs for almost any application.

There is a text file named Readme.txt on the supplied disk that contains any late-breaking information about NS Basic/CE, including updates to the Handbook. Please read it before installing NS Basic/CE.

If you'd like to get started using NS Basic/CE right away, then read the Installation section, and then turn to the Getting Started With NS Basic/CE chapter.

Sample programs are provided with NS Basic/CE for you to study and use. You can tailor these sample programs to your particular needs. The installer will put these examples in a folder titled "BASIC Samples" in the target installation folder.

You should be somewhat familiar with the basics of operating a Windows CE device before you start using this Handbook. You should know about starting applications, using the stylus and other Windows/CE features. If you are not comfortable with these terms, review the Windows CE Handbook.

A basic understanding of operating a desktop computer (Windows 98/ME/NT/2K/XP/Vista) is needed to install the NS Basic/CE software.

1.1 What is BASIC?

BASIC has been around for almost 40 years. Over that period, hundreds of interpreters and compilers for BASIC have been developed, and a mountain of application code has been written. Many books continue to be published about the language. BASIC Special Interest Groups exist in a number of forms.

BASIC is somehow good for the soul. As new waves of languages come and go, BASIC still runs almost everywhere: without standards, it adapts to new environments easily and keeps pace with the fancy new languages. The ones that come and go.

Everyone, even Bill Gates, started with BASIC. Somehow, we all keep coming home to it over and over again. It's still the best language for quick programs and simple applications. BASIC interpreters, especially simple ones, can have great charm.

The computer hardware that BASIC is programmed on has turned full circle since the days it was developed. The powerful language to which only the computer scientists and mainframe programmers had access to can now be run on a hand held device.

1.1.1 NS Basic/CE

NS Basic/CE for Windows CE is a real programming language. It implements all the commonly used BASIC statements in a straightforward manner, and has a number of powerful extensions.

NS Basic Corporation maintains a World Wide Web page at <http://www.nsbasic.com>. Use your Web browser to check this site for important announcements, technical information, and example NS Basic/CE programs.

1.1.2 NS Basic/CE and Windows CE

When you bought your Windows CE device, you probably thought that you'd be able to replace many of your paper-and-pencil tasks with it. You probably also hoped it would be able to function as a small programmable computer. NS Basic/CE has been designed for this purpose. Using it, you'll be able to create the applications you need, in a language that is easy to use, right on your Handheld PC.

NS Basic/CE can also be used for general purpose programming. Any program that can be written in BASIC can be written in NS Basic/CE. You can create customized databases, perform complex calculations, or even write games. What sets NS Basic/CE apart is its accessibility. You don't need to learn a complex new language just to take advantage of the powerful features built into your H/PC.

1.2 System Requirements

In order to install NS Basic/CE you will need a Windows CE device, a desktop computer running Microsoft ActiveSync, and a cable that can be used to connect the Windows CE device to the desktop computer. Check the Readme.htm file included with your product for specific information on devices supported and software required.

1.3 Installation

NS Basic/CE is supplied on a CD. You must install it onto your Windows CE device using Microsoft ActiveSync. Check the Readme.htm file included with your product for installation instructions.

You should also pay a visit to our website, where we post the latest news on NS Basic/CE. Check the Release Notes for updates, the Tech Notes section for the latest info, and the Downloads section for additional sample code, updaters and the Runtime modules.

Our website is <http://www.nsbasic.com>.

1.3.1 Entering Your Serial Number

Before you will be able to use NS Basic/CE, you will have to enter your serial number. On your device, run the BASIC Installer program. On your desktop, use Register under the Help menu. Your serial number is on the back of your handbook. Check the Readme.htm file for additional information on installation.

If you received a registration card, please fill the card out now, and send it to the address printed on the card, or register on line at our web site.

2. NS Basic/CE Concepts

2.1 Conventions Used in this Handbook

The following notation conventions are used in this Handbook:

KEYWORDS

Capital letters indicate NS Basic/CE keywords and other text that must be typed exactly as shown. For the purposes of this manual, uppercase text indicates a required part of the Statement syntax. NS Basic/CE is case-insensitive: keywords are accepted with either uppercase letters, lowercase letters, or any mixture of the two. A keyword such as PRINT may be entered into your programs as print, Print, or PRINT.

placeholders

Italic text indicates a placeholder for types of information that you must supply. In the following Statement, expression is italicized to show that the EXECUTE statement requires an expression:

EXECUTE *expression*

Example

This Monaco typeface indicates example program code and information that is printed on your NS Basic/CE screen. The following example shows a line from an NS Basic/CE program:

```
PRINT "Hello World!"
```

[Optional]

Brackets indicate that the enclosed items are optional. In the following example, brackets are used to show that entering a second item to display on the screen is optional for the PRINT statement:

```
PRINT expression1 [ ,expression2 ]
```

Both of these PRINT statements are legal, since PRINT accepts up to 20 expressions:

```
PRINT "Hello"
```

```
PRINT "Hello", "World"
```

|

The vertical bar indicates that the items are mutually exclusive. In the following example the bar indicates that the LEN function can either be used with a string or a variable name:

LEN(*string* | *variable*)

2.2 The Elements of an NS Basic/CE Program

A program in NS Basic/CE is a set of Statements. Each NS Basic/CE program line may consist of the following elements:

KEYWORD arguments 'comment

A KEYWORD is a word from the language that NS Basic/CE understands. Examples are PRINT, INPUTBOX and IF. The Statement and its arguments determine what action (if any) will be taken by NS Basic/CE when the line is executed.

Any text following ' on a line is a comment, and is ignored by NS Basic/CE.

2.2.1 Multi-Line Statements

Each NS Basic/CE statement normally ends at the end of the statement line. If you have a very complex statement the line can be very long. This can make your programs difficult to read. You may split long statements by using the line-continuation sequence, a space followed by an underscore, (`_`) at the end of the line. NS Basic/CE combines the current line with the next line if the current line ends in (`_`). Here is an example of line continuation:

```
PRINT "Long statements are no problem for" _  
      & " NS Basic/CE"
```

2.2.2 Multiple Statements per Line

By using a `:` character as a separator, you can put more than one statement on a line:

```
a=1: b=2: c=3
```

2.2.3 Literals, Data Types and Variables

Literals are literal values you use in your programs. You use them all the time: to set the initial value of a variable, to establish the starting and ending values of a FOR...NEXT loop, and so on. When you define a constant using the CONST Statement, the name of your constant can be treated as a literal. You cannot change the value of a literal.

Variables are the named holders of your data. A variable's value may be changed as needed. All variables are of data type variant, with a variety of subtypes.

2.2.4 Numeric Data Types

There are several subtypes for Numeric data, but they all share the same behavior. You can generally mix and match among the types of numeric data without difficulty.

Subtype	Size	Range	Literal
Byte	8 bits	0 to 255	2
Int	16 bits	-32,768 to 32,767	100
Long	32 bits	-2,147,483,648 to 2,147,483,647	500000
Single	32 bits	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$	1.0e100
Double	64 bits	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$	1.0e1000
Currency	32 bits	-9.2×10^{14} to 9.2×10^{14}	1000000.00
Hex	32 bits		&h000000F

2.2.5 Boolean Data Type

Booleans consists of two values: TRUE and FALSE. This data type is used with the IF Statement. It tests the Boolean value of an expression and selects the THEN Statement if it is TRUE, or the ELSE Statement if it is FALSE. The numeric value for TRUE is -1, and the numeric value for FALSE is 0.

2.2.6 Color Data Type

Colors are unsigned, long integers with values from 0 to 16,777,215. A color value is created by mixing amounts of red, green, and blue that vary from 0 to 255, where 0 is dark and 255 is bright. The mixing formula is:

$$\text{color} = \text{red} + (\text{green} * 256) + (\text{blue} * 65536)$$

Shades of black and white are created when equal amounts of red, green, and blue are used. See Appendix B for a list of common color constants.

Color Name	Red, Green, Blue	Color Value
Black	0, 0, 0	0, vbBLACK
Dark gray	128, 128, 128	8,421,504
Light gray	192, 192, 192	12,632,256
White	255, 255, 255	16,777,215, vbWHITE

2.2.7 String Data Type

Strings consist of a series of characters. A string can be approximately 2 billion characters long. There are a number of functions that manipulate strings. The concatenation operator (&) is used to join the string representations of two variables together. A string literal is enclosed in quotation marks:

```
"This is a string literal"
```

2.2.8 Array Data Type

Arrays are containers. They are lists of values stored with a single name. Each element in the array is referred to by including a number in parentheses after the variable name. The first element of an array is always zero (0), and each array can have many elements. ARR(2) refers to the third element in the ARR array. Each element in an array can be of any subtype.

2.2.9 Variable Names

A variable is a name that holds a value. The name consists of a sequence of alphabetic and numeric characters, and the underscore (_) characters. There is no limit to the length of a variable name in NS Basic/CE, and every character in the name is significant. We tell you this because in some older BASICs you could only use short names. Variable names are not case sensitive, and spaces and other special characters may not be used. Variable names must start with a letter. NS Basic/CE constants may not be used as variable names. For a complete list of constants, see Appendix B.

The following list shows some variable names that are allowed by NS Basic/CE:

```
text
LLAMAS      'same as llamas or Llamas
Jupiter
WlSpec
SouthPark
```

And some that are not allowed:

```
ltable      'starts with a number
X&Ycords    'uses special character &
first counter 'has a space
%correct    'does not start with a letter
print       'NS Basic/CE keyword
```

2.3 Expressions and Operators

An expression is a literal, variable, formula or FUNCTION procedure call that has a value. Here are some examples of expressions:

```
6/3          'result is 2
```



```
5+6/3          'result is 7
"This " & "that" 'result is "This that"
```

A string expression can be a string literal, a string variable, or it may combine string literals, string variables and substrings to produce a single string value. Similarly, a numeric expression can be a numeric constant, a numeric variable, or a function/variable that produces a single numeric value.

2.3.1 Arithmetic Operators

NS Basic/CE allows the following arithmetic operators in this descending order of priority:

()		Parenthesis
^		Exponents
*	/ \	Multiplication and Division
+	-	Addition and Subtraction

Parenthesis can be used to change the order of evaluation.

```
PRINT 2 + 3 * 4
PRINT (2 + 3) * 4
```

Displays

```
14
20
```

NS Basic/CE supports floating-point arithmetic. The MOD function may be used to find the remainder of a division. The backslash operator (\) is used for integer (whole number) division.

Arithmetic operators can only be used with numeric expressions. They may not be used with strings.

2.3.2 Relational Operators

Relational operators compare two values and return a Boolean value of TRUE or FALSE. This result can be used to change the flow of a program. Relational operators have a lower priority than arithmetic operators. The relational operators are:

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

In the SET statement, the equal sign is used to assign a value to a variable, not as a relational operator.

2.3.3 Boolean Operators

Boolean operators tie expressions together, returning a TRUE or FALSE answer. Arithmetic and relational operators are evaluated before Boolean operators. The NOT operator applies to one expression, while the other operators require two expressions.

The Boolean operators are:

AND	Returns TRUE if the two expressions are both TRUE.
EQV	Returns TRUE if both expressions are TRUE or both expressions are FALSE.
IMP	Returns TRUE if the first expression implies the second expression.
OR	Returns TRUE if either expression or both of the expressions are TRUE.
NOT	Returns TRUE if the expression is FALSE or returns FALSE if it is TRUE.
XOR	Returns TRUE if the neither of the expressions matches the other.

Boolean operators can be used with any expression that returns a Boolean value..

2.4 FUNCTION and SUB Procedures

Procedures are blocks of program statements that can be conditionally and repeatedly executed ("called") from other statements in a program, with optional values (arguments) passed in. When a procedure is called, it behaves like the Functions and Statements that make up the NS Basic/CE language. FUNCTION procedures return a value, which can be stored in a variable or used in another expression. SUB procedures execute without returning a value. If a FUNCTION procedure is called and the return value is not used, NS Basic/CE executes it as a SUB procedure.

To pass in multiple arguments to a FUNCTION procedure, use a comma-separated list, enclosed in parenthesis. To pass in multiple arguments to a SUB procedure, use a comma-separated list with no parenthesis. When a single argument is passed in to either a FUNCTION or a SUB procedure, parenthesis may be used; NS Basic/CE uses parenthesis around a single argument to evaluate an expression, not to denote an argument list.

2.5 Projects, Modules, Forms, Objects and Controls

Controls and Objects

A control is an executable module that can be accessed by other programs. Each instance of a control in your program is called an Object. To add an object to a program, use the ADDOBJECT Statement. Objects have Properties which can

be queried and set, Methods which can be called as FUNCTION or SUB procedures, and Events which can be triggered by user actions, the operating system, or other programs.

A number of controls are installed with NS Basic/CE. These include a checkbox, combobox, textbox and other common objects. These are sometimes referred to as intrinsic objects as they do not rely on another file being present, but are contained directly in the NS Basic Runtime.

A second group of controls, called the Standard Controls, is included with each NS Basic installation. These are ActiveX controls supplied by Microsoft, such as PictureBox, Grid and FileSystem. Each is contained in a separate file of type dll.

Third party ActiveX controls can also be used with NS Basic/CE. A number of these are in NS Basic's Big Red Toolbox. Look at Tech Note 1 or do a search on the web to find more.

All installations of NS Basic/CE include the PictureBox control. The background of the output window, known as the Output object, is a PictureBox. In addition to the Output object, an Err object is created automatically in every running program, for optional error handling.

Object Events

An event is a call that an object makes to your program, as a result of an action in the object. For instance, if you have a commandbutton in your program, an event will take place when the user taps the button. If your button's name is "MyButton", NS Basic will call the function MyButton_Click() in your program. If you do not have such a function, the event is ignored.

Another example would be if you were doing serial communications. Incoming data would cause the OnComm event to be sent. If you called your communications object "Comm", it would try to call the function Comm_OnComm().

Forms

NS Basic/CE's implementation of forms is simpler than that on the desktop. While the underlying engine does not have the concept of a form, there is a technique for providing the same functionality.

Once an object is created, there is no way to uncreate it. In Windows CE, this is not an issue: the overhead of each object is low enough that it is no problem keeping all your objects around until your program ends.

A Form is therefore just a group of objects that are shown and hidden at the same time. You can keep references to each of the objects of a form in an array. A form can be

hidden by going through each of the elements of the array and hiding the object it refers to .

While you can do this yourself, if you are using the Desktop IDE or the Visual Designer, NS Basic/CE will take care of creating the necessary code for creating, hiding and showing forms.

Forms are based on the PictureBox object, and have the same properties.

Form Methods

Each form has a Hide and a Show method. If your form is named "Form1", the following code in your program will hide it:

```
Form1_Hide
```

To show a form, use Form1_Show. The first time a form is shown, NS Basic will create the objects for the form. While the form is being created, the variable Form1_temp is not used (assuming your form is called Form1). When all the objects are created, isObject(form1_temp) is true. You can use this variable to check if a form is fully created before responding to events that one of the objects on the form might generate.

Form Events

If your program was created using the Desktop IDE, a form load event is sent to your program. Assuming your form is called Form1, the subroutine Form1_Load() will be called just after the objects are created by Form1_Show. Likewise, when you close a form, a Form1_Unload event will be sent.

Program Properties and Events

NS Basic programs run within a PictureBox object called the Output object. See the PictureBox control for a list of properties and events that are supported.

```
REM Turn application background to blue
Output.BackColor=vbBlue
```

Program Events

When a program is closed (or minimized on a Pocket PC), a Output_Close event is sent to your program. This event is sent to all programs whether created by the Desktop IDE or not. This feature is especially useful for Pocket PC devices. Microsoft's Pocket PC user interface guidelines specify that the button on the top right of the application, which closes the app on other versions of Windows, should just minimize the app and keep it running. To override this behavior in your app, add the following code to your application:

```
REM Close app when minimized
Sub Output_Close()
    Bye
End Sub
```

You can also use the ShowOKButton statement to change this behavior.

If the output window is resized, for example if the screen is rotated, an Output_Size event is sent to your program.

Modules

A Module is a .txt, .cod or .bas file that is included in the project. It can be used to break up your code into smaller pieces so your project is more manageable. Modules can also be used in more than one project, making it possible to share code during development. At runtime, all of a project's modules are merged into a single running program. Any code in modules is placed at the end of the main project.

Hi Res Aware

Some Windows Mobile devices have a screen size larger than 240x320. Devices with VGA screens are 480 x 640, for example. NS Basic/CE programs running on devices with different screen sizes have always looked exactly the same: the 240x320 coordinates are mapped on the VGA screen so everything is still in the same place and the screen completely filled.

However, it is possible to use the extra pixels. By setting Hi Res Aware to True in Project Properties, your program will

treat the screen as it actually is 480 x 640. This will let you create smaller objects and smaller text.

If you want your project to be Hi Res aware, turn on Hi Res Aware in Project Properties. In your program, check output.height and output.width in your Form_Load routine. Adjust object bounds and font sizes if necessary.

Icons

To replace the default NS Basic icon in your app, create a .ico file and put its name into the Icon property in Project Properties. Which sizes are needed depending on what devices you are supporting. If you want to support all devices, you will need at least the following: 16x16, 21x21, 32x32, 43x43 and 64x64. A good free tool for making .ico files is IcoFX.

3. Programming On the Device

3.1 Overview

NS Basic/CE 's development environment for programming on the device has two different views, the Program Editor and the Output Window. It also has a tool called the Visual Designer, a graphic way of creating code for the Program Editor.

3.2 Program Editor

The program editor is where you write your programs. It has a white background with a vertical scrollbar, and the toolbar that controls NS Basic/CE. The menu section of the toolbar is outlined below, in Table 1

Table 1: NS Basic/CE Menus






Menu	Command	Description
File	New	New program
	Open	Load a program
	Save	Save a program
	Save As...	Save a program with a new name
	Encryption ON/OFF	Sets the encryption status of the saved program. Only the encryptor's copy of NS Basic/CE may see the program's source code.
	1...4	Last four programs
	Exit	Quit NS Basic/CE



Table 1: NS Basic/CE Menus

Menu	Command	Description
Edit	Undo	Undo/Redo last edit
	Cut	Cut selected text
	Copy	Copy selected text
	Paste	Paste clipboard
	Select All...	Select all program text
	Find...	Perform search
	Find Again	Repeat last search
	Find All...	Find all matches
	Goto Line...	Display a line in editor.
	Overview	List all procedures
Tools	Format	Reformat a program
	Run	Execute a program
	...Run	Run the program
	..Execute Function	Restarts at a function
	..Trace	Trace execution
	...Step	Statement at a time
	Execute Code...	Enter and execute a
	Show Variable...	Display variable value
	Stats...	File size data
	Visual Designer	Visual Forms Editor
Help	Help Topics...	Help browser
	About NS Basic/CE...	Title, version, contact

Tool buttons are to the right of the Menu Items.

Table 2: Tool Buttons

Button	Menu Selection	Keyboard Shortcut
	"File->New"	Ctrl+N
	"File->Save"	Ctrl+S
	"File->Open"	Ctrl+O
	"Edit->Cut"	Ctrl+X
	"Edit->Copy"	Ctrl+C

Button	Menu Selection	Keyboard Shortcut
	"Edit->Paste"	Ctrl+V
	"Tools->Run"	Ctrl+R

Two other buttons are located in the upper right corner of the program editor window. The help button opens the help browser with NS Basic/CE online help and is a shortcut for "Help->Help Topics..."



The close button exits NS Basic/CE and is a shortcut for "File->Exit".



3.3 Output Window

The output window is displayed while your program is executing, and it is where any input or output takes place. It has a light gray background with a close button in the upper right corner (depending on the version of Windows CE). A small line in the upper left corner of the window shows where a program can add a custom menu of its own with the SETMENU statement. The close button closes the output window; if the program was run from inside the program editor, control returns to the editor, otherwise NS Basic/CE exits.

3.4 Visual Designer

The Visual Designer is part of NB Basic/CE, a full featured BASIC development environment. It allows you to visually create forms on your Windows CE device. It is started from the NS Basic/CE environment.

To add an object to a form, select the object you want from the Objects Menu. It will get created in the top left corner of your form. You may drag it and resize to set the bounds. Double tap on it, and the Property Editor for the object will appear. You can customize the values for any of the properties of the object.

When you close the Designer, your specifications are turned into NS Basic/CE code and you are returned to the normal NS Basic/CE editing window. Your code can be run immediately.

You can reenter the Visual Designer at any time to make further changes to your format, even if you have modified other parts of your code.

WARNING! If you modify the code that has been generated by Visual Designer, you may not be able to edit it further in Visual Designer. You may also lose your changes if you generate again.

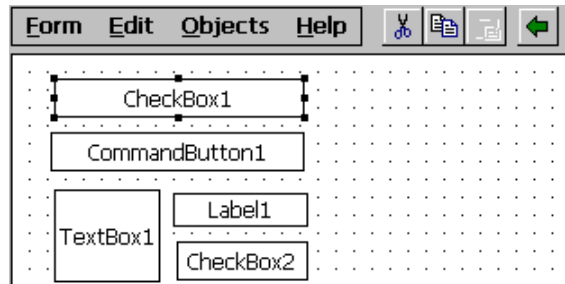
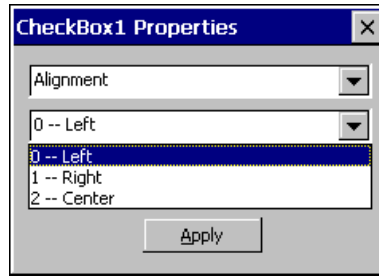


Table 3: Visual Designer Menus

Form	Add Form	Add a new form to the program.
	Delete	Delete a form.
	Set Default	Make current form the default.
	Exit	Exit back to NS Basic Editor.
Edit	Undo	
	Cut	
	Copy	
	Paste	
	Clear	
	Select Object	Bring up list of objects to select.
	Properties	Edit properties of the current object. If no object is selected, the properties of the form itself. See Property Editor.
Object	Add Menu	Add menu to current form. See Menu Editor.
	Edit Menu	Edit menu on current form. See Menu Editor.
	Show Program	Display the program's code
	Snap to Grid	Move object will snap to grid
	Grid Size...	Set Grid size to 4-20 pixels
	CheckBox	Add a checkbox object.
	ComboBox	Add a ComboBox object.
	CommandButton	Add a CommandButton object.
	Date	Add a Date picker object.
	Label	Add a Label object.
	ListBox	Add a ListBox object.
	OptionButton	Add a OptionButton object.
	TextBox	Add a TextBox object.
	Time	Add a Time picker object.
Help	Help Topics	
	About VNSB	

3.4.1 Property Editor

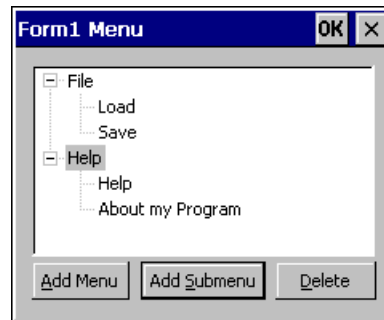
Each object type has its own list of properties. The upper



combobox is used to select the property to be edited. The lower box will give a list of allowable values or other input field, depending on the requirements of the property.

3.4.2 Menu Editor

Menus can be set up with multiple layers of submenus. For each menu item, you can specify the Caption, Menu Key and



Accelerator.

3.4.3 Notes

1. ComboBox and ListBox have a special property named List. Each line entered into the edit box becomes a separate choice. If you end the line with a single quote (') character, it is taken as an expression to be evaluated.

3.5 The NS Basic/CE Programming Environment

NS Basic/CE provides a full featured BASIC programming environment for Windows CE. In order to introduce you to these features, an example program will be developed in NS Basic/CE. Each step in the process will introduce features of the environment. Start NS Basic/CE and follow along!

3.5.1 Creating a Program

When you create a new program, the output window must be closed and the program editor must be empty. If the output window is open, tap the close button in the upper right corner of the window, then select "New" from the "File" menu.

The example program will be simple. It will compute the total cost of a sale by adding the cost of the individual items and any applicable tax. The program will be created with a couple of errors so we can practice debugging.

Enter the program shown below:

```
REM Sales Calculator
OPTION EXPLICIT
'Declare Constants
APPTITLE = "Sales Calculator"
'Declare variables
DIM SubTotal, TaxRate, ItemCost, Item

'Initialize variables
SubTotal = 0
Item = 1

'Get item costs
DO
    ItemCost = INPUTBOX("Item " & Item & _
        " cost", APPTITLE, 0)
    IF LEN(ItemCost) AND NOT ItemCost = 0 THEN
        PRINT FORMATCURRENCY(ItemCost)
        SubTotal = SubTotal + ItemCost
        Item = Item + 1
    END IF
LOOP WHILE LEN(ItemCost) AND NOT ItemCost = 0
IF SubTotal = 0 THEN BYE
```

```
'Output
PRINT FORMATCURRENCY(SubTotal), , "Sub
Total"
TaxRate = INPUTBOX("Tax Rate (10% = 10)", _
    APPTITLE, 0)
PRINT FORMATCURRENCY(SubTotal*TaxRate), , _
    "Tax"
SubTotal = SubTotal * (1 + TaxRate)
PRINT FORMATCURRENCY(SubTotal), , "Total"
```

NOTE: The InputBox statement in the above code will not work on Pocket PC devices without some extra code. Please see the Readme.htm document for additional information.

3.5.2 Editing a Program

The program editor works like a standard text editor. A blinking cursor marks the current insertion point, where any new text will be added. The cursor can be moved with the arrow keys, page up and page down, home and end, and by tapping a stylus on a touch sensitive screen.

Cut, Copy, and Paste allow you to move and remove selected sections of text. To make a selection either tap on the screen and drag with your stylus, or position the cursor at one end of the desired area and hold down the shift key while you navigate to the other end with the keyboard. "Cut", "Copy", and "Paste" are in the "Edit" menu.

To undo your most recent change to a program, choose "Undo" from the "Edit" menu. There is a maximum size of 1,048,575 bytes in this editor: larger programs may be edited using the Desktop Editor.

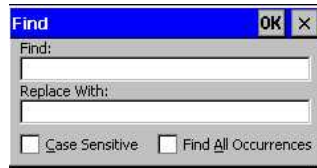
3.5.3 Formatting a program

To make your program easier to read, use spaces or tabs to indent related statements. This is usually done with statements inside procedures and loops. The default tabwidth and indentation in NS Basic/CE is two spaces. To automatically reformat an entire program, choose "Format" from the "Tools" menu.

3.5.4 Finding and Replacing

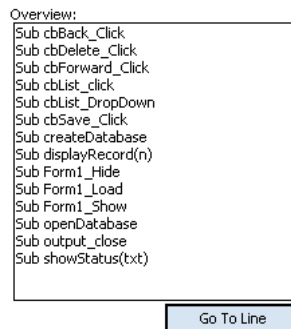
The body of your program can be searched in one of two ways. Using Find you can find the nearest existing match to the current cursor location; Find Next will perform the most recent Find, and can be used with Find to locate successive matches of the search string. If a match is found this way, it will be hilited and the program editor will scroll down (if necessary) to display it. Any search pattern can be matched with or without case sensitivity. Find All Occurrences allows you to enter a search string, and returns a list of every line in

the current program that contains a match. "Find" and "Find Next" are in the "Edit" menu.



3.5.5 Overview

The Overview command displays a list of all FUNCTION and SUB procedures. The procedures are first sorted by type (FUNCTION or SUB), then they are sorted alphabetically. The Overview display can also be used to navigate to selected procedures. Select a procedure and click on the "Goto Line" button, or double tap a procedure to close the Overview display and show the procedure in the program editor.



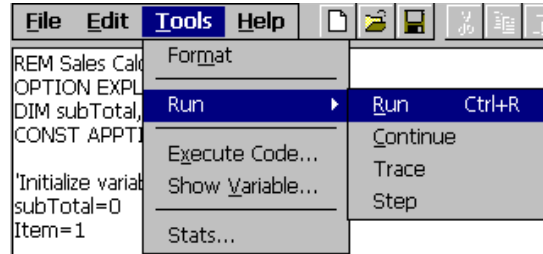
3.5.6 Running a Program

To execute your program, choose "Run" from the "Tools" menu or the run icon.

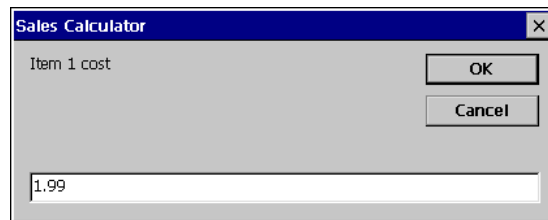


Let's run our example program with three items costing \$1.99, \$2.99, \$3.99, and a tax rate of 6%.

Running the program...



Entering the first item...



Program output...

Program	
\$1.99	
\$2.99	
\$3.99	
\$8.97	Sub Total
\$53.82	Tax
\$62.79	Total

3.5.7 Debugging a Program

The balance we computed was far higher than the cost the individual items and the proper tax amount. It seems we've got a bug in our program. There are three types of errors

possible in NS Basic/CE: compile-time errors, run-time errors, and logic errors.

A compile-time error occurs when your program's syntax is incorrect. Examples of compile-time errors include: misspelled keywords, calling a SUB procedure with parentheses, and required arguments that are missing.

A run-time error occurs when a program is executing and NS Basic/CE encounters a statement it either can't perform or doesn't understand. Examples of statements that can't be performed: dividing by zero, multiplying strings, and referencing an object that hasn't been added. Examples of statements that can't be understood: statements with keyword typographical errors, multi-line statements that don't use the continuation sequence space-underscore (_), and comments that don't begin with REM or an apostrophe (').

A logic error occurs when your program's algorithm is incorrect, and the program executes without error. The error (or errors) can only be detected by wrong or unexpected results. Once you have become familiar with programming in NS Basic/CE, this may be the only kind of error your programs make. Unlike a run-time error, NS Basic/CE cannot detect logic errors.

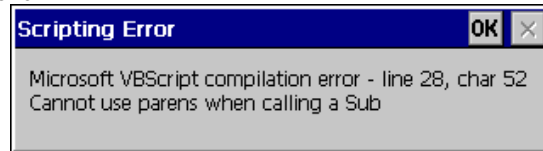
Logic errors are common and our error is a logic error. We did not receive any error messages when the program ran, but the total displayed was too high. We'll have to debug our program!

The easiest way to debug a logic error is to get more information from the program during execution. The two simplest ways to get output from a program are the PRINT statement and the MSGBOX function.

Looking at the output, we see that the correct Sub Total is calculated, so we know that our program is computing the tax incorrectly. Let's insert a call to MSGBOX with the value of the tax rate and the multiplier that is applied to the Sub Total. Add the following line to the end of the program

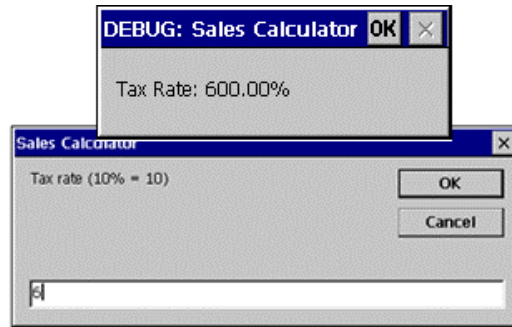
```
MSGBOX("Tax Rate: " _  
    & FORMATPERCENT(TaxRate), 0, "DEBUG: " _  
    & APPTITLE)
```

When we try to run the program again we get the following error:



This is a compile-time error that has occurred from improper syntax. When the return value from a FUNCTION procedure, in this case MSGBOX, isn't assigned to a variable, it is treated as a SUB procedure. SUB procedures cannot be called with their arguments in parentheses, so we will have to remove them from the line we just added.

Once this is done, the program executes without error and

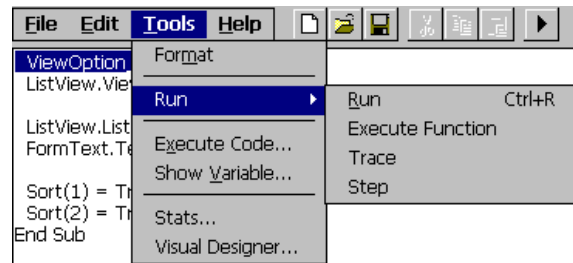


our debug output shows the following:

From this we can tell that the Tax Rate isn't being handled properly, it needs to be divided by 100. After changing the Output section of the program to match the section below, the program runs without error and calculates properly. In addition, we've added a bit of extra information by reporting the Tax Rate in the program's normal output.

```
'Output
PRINT FORMATCURRENCY(SubTotal), , "Sub
Total"
TaxRate = INPUTBOX("Tax Rate (10% = 10)", _
    APPTITLE, 0) / 100
PRINT FORMATCURRENCY(SubTotal * TaxRate), , _
    "Tax (" & FORMATPERCENT(TaxRate) & ")"
SubTotal = SubTotal * (1 + TaxRate)
PRINT FORMATCURRENCY(SubTotal), , "Total"
```

3.5.8 BREAK, Execute Function, Trace and Step

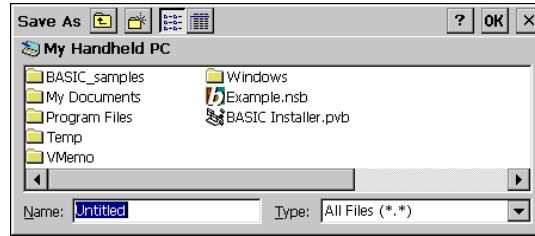


These tools combine to give you powerful debugging capability. Use the Break statement to set a breakpoint in your program. If you would like to see each statement displayed as it is executed, start your program using Trace instead of Run. If you choose Step instead of RUN, a Break will automatically be executed after each statement, allowing you to step through your program one statement at a time. Trace and Step will not work well with an `OPTION EXPLICIT` statement unless it is the first line of the program.

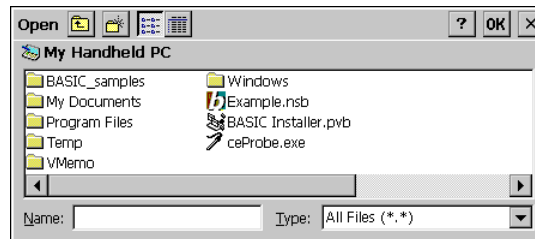
Execute Function will bring up a list of functions and subroutines. Click on one to execute it directly. This is useful in debugging a function directly.

3.5.9 Saving and Loading a Program

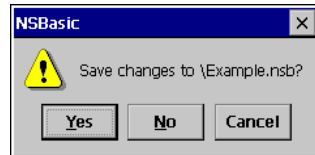
When you create a new program (by selecting "File->New"), it only exists in the program editor. If you quit NS Basic/CE or load another program and wish to save your current program, you must select "Save" from the "File" menu. This will open a dialog box prompting you to name the file that your program is stored in. Let's save our example program with the name SalesCalculator.nsb.



When you would like to edit or run a program you have already saved, you must select "Open" from the "File" menu. Let's open the example program we just saved.



Whenever NS Basic/CE is going to close a program that has been modified, it will prompt you to see if you would like to save the changes.



3.5.10 Saving and Loading Programs as Text Files

If you want to use your code in another application or a desktop system, you can save it as a text file. In the Save As dialog, add .txt after the Name. You can also load text files.

4. Programming on the Desktop

NS Basic/CE allows you to develop programs on your desktop that will run on a Windows CE device. You can view the program running on your device from the desktop using Virtual CE.

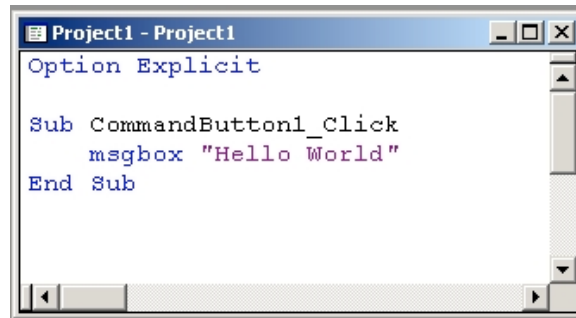
4.1 Menu Options

File	
New Project	Open a new project. The current one is closed first.
Open Project...	Bring up the Open Dialog for Existing and Recent projects. See Open Dialog for more information.
Save Project	Save the project. If project is new, a name will be requested.
Save Project As...	Save the project under a name you select. Files can be saved in .nsb or .txt format.
Print Setup...	The standard Windows Print Setup.
Print...	Print the project.
Recent Files	A list of recent projects you can open.
Exit	Close the current project and exit the IDE.
Edit	
Undo	Undoes the last edit in the Code Window. Multiple undoes are supported.
Redo	Does the last edit in the Code Windows again.
Cut	Cut selected code or object. Objects are cut (and copied) in standard text format, so you can view and edit them in other applications.
Copy	Copy selected code or object.
Paste	Paste code or object.

Delete	Delete the currently selected code or object.
Find...	Find searches the code for a string. It can search the entire project or just the currently selected code. Options exist for finding whole words and for searching as case sensitive.
Replace...	Replace is similar to Find, but can be use to replace one or all occurrences of a string.
Go To Line	Scrolls the cursor to the specified line number in the topmost code window.
View	
Project Explorer	Display or hide the Project Explorer Window.
Properties Window	Display or hide the Properties Window.
Toolbox	Display or hide the Toolbox. The Toolbox displays a list of available objects.
Toolbar	Display or hide the Toolbar (just under the menus)
Status Bar	Display or hide the Status Bar (at the bottom of the IDE).
Refresh	Redraw all Windows.
Project	
Add Form	Add a new form to the project. A form is a collection of objects that are displayed by a program at the same time. A program may have any number of forms.
Format	
Size to Grid	Resizes the selected object to align with the grid pattern. The size of the grid pattern can be selected in Tools...Option...CE Screen.
Center in Form	Moves the selected object so it is centered (horizontally or vertically) on the form.
Run	
Start	Starts running the program. The program will be copied to the device. Depending on the setting in Tools...Options...Start, it will be started automatically.

Active Sync Program	Copy current program to device using ActiveSync.
Installers...	Brings up a list of installers for devices that are in the Installers folder.
Start Virtual CE	Starts Virtual CE. Your device must be connected by ActiveSync for this you work.
Tools	
Menu Editor	Start the Menu Editor.
Options...	Start the Options Window.
Window	
Cascade	Organize the open CE Screen and Code Windows to cascade neatly on the screen.
Tile Horizontal	Reposition and resize open CE Screen and Code Windows one after the other down the screen.
Tile Vertical	Reposition and resize open CE Screen and Code Windows one after the other across the screen.
Help	
Register...	Enter your serial number here to activate your copy. Your serial number is on the back of your Handbook.
Help	Bring up the Reference chapter of the Handbook on the screen.
NS Basic Website	Go to NS Basic's website.
Tech Notes	Display Tech Notes with additional information not in the Handbook.
Big Red Toolbox	Display information on additional ActiveX controls that are available for Ns Basic users.
About NS Basic/CE	The standard NS Basic About screen. Use this to check your current version of NS Basic.

4.2 Code Window



The code for your program is entered in the Code Window. All of your code is in a single code module, but if you click on an object in the CE Screen, you will be positioned automatically to the proper position in the code.

The text is colored depending on its type:

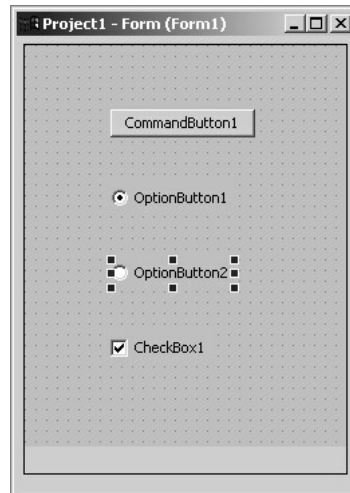
Black	Program text
Blue	NS Basic/CE keywords
Green	Comments
Orange	Operators
Purple	Strings
Yellow	Highlighted line

Going into Options...Editor brings up a few handy options. The Properties selection allows a wide range of features to be customized, including text coloring, tabs, font, keyboard shortcuts and window appearance.

Statements can be made longer than one line by using the "_" character at the end of a line.

Cut, Paste, Delete and other similar functions can be used from the Menu, the Toolbar or right clicking.

4.3 CE Screen



The CE Screen is a mock up of how your application will look when running on the device. The background grid that displays by default does not show up on the actual device, but makes it easier to lay out your form. You can change the spacing of the grid points or hide them by changing the setting in Options.

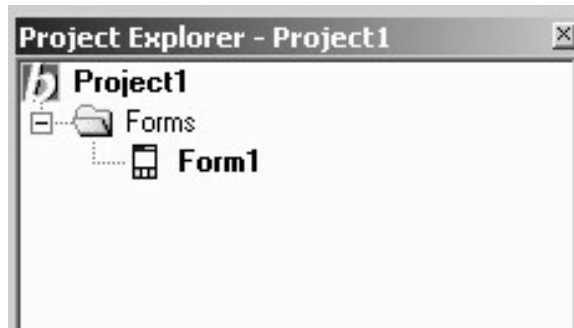
To add an object to the CE Screen, select it from the Toolbox and click in the CE Screen at the position you want the top left corner of your new object. The new object will display.

To select an object, click on it. The functions under the Format menu can be used to position objects relative to the screen. When an object is selected, its properties are displayed in the Properties Window. When you edit the properties there, the changes are reflected immediately on the CE Screen. To edit the code for an object, double click on it and a Code Window will open.

Clicking outside any of the objects will bring up the Properties Window for the form. To rename or delete an object, right click on it on the CE Screen.

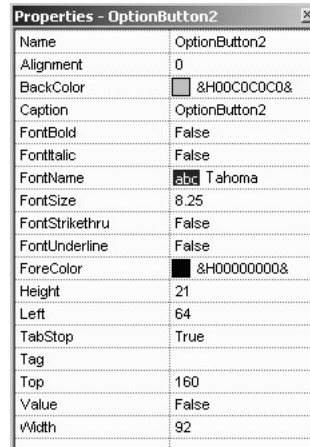
Double clicking on the form will bring up the Form_Load subroutine for editing. You will need to add the Form_Unload subroutine yourself: there is no way to bring it up automatically.




4.4 Project Explorer



The Project Explorer window is used to navigate through your project. You can use it to see what forms are part of the project. By selecting one, you'll see the layout of the form on the CE Screen and a list of the form's properties and values in the Properties Window.

4.5 Properties Window

A screenshot of the 'Properties - OptionButton2' window. It displays a list of properties and their values for a control named 'OptionButton2'. The properties include Name, Alignment, BackColor, Caption, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Left, TabStop, Tag, Top, Value, and Width. Most values are standard defaults, such as 'False' for bold and italic, 'Tahoma' for font name, and '8.25' for font size. The BackColor and ForeColor are represented by small color swatches and hex codes. The window has a standard title bar with a close button.

Name	OptionButton2
Alignment	0
BackColor	 &H00C0C0C0&
Caption	OptionButton2
FontBold	False
FontItalic	False
FontName	 Tahoma
FontSize	8.25
FontStrikethru	False
FontUnderline	False
ForeColor	 &H00000000&
Height	21
Left	64
TabStop	True
Tag	
Top	160
Value	False
Width	92

The Properties Window shows the values of the properties of the project, its forms and the objects on the forms. The values can be edited, and where possible, the CE Screen will be updated to reflect the changes.

For an explanation of the different properties, see "Properties" in the Reference chapter of this Handbook.

4.6 Toolbox



The Toolbox contains icons that represent the standard controls that can be placed on a form.. By holding the cursor over each icon for a few seconds, you can see its description.

The ActiveX icon is a special case. It allows you to use other controls not shown in the Toolbox. Simple click on the ActiveX icon and identify the control you wish to use. It will then be added to the project normally.

For more information on the controls, see the AddObject statement in the Reference part of the Handbook. Intrinsic objects are documented in the Reference section. The more advanced objects are documented in the Tech Notes.

Not all ActiveX controls can be added to the Toolbox. They need to have a desktop version that has all properties correctly defined by the author. However, ActiveX controls can be added to your project using the AddObject statement, bypassing the ToolBox. In this case, properties are edited in your code instead of the Properties Window.

To change the name of an object after it is created, right click on the object and select Rename from the menu that pops up.

4.7 Toolbar



The toolbar contains icons for common operations. These can all be found in the menu as well.

New
Open
Save
Print
Undo
Redo
Cut
Copy
Paste
Delete
Search
Add Form
Start
Properties
Menu Editor

4.8 Menu Editor

The screenshot shows a window titled "Menu Editor - Form1". It contains three input fields: "Caption:" with an empty text box, "Name:" with an empty text box, and "Shortcut:" with a dropdown menu showing "(None)". To the right of these fields are "OK" and "Cancel" buttons. Below the fields is a row of buttons: a left arrow, a right arrow, an up arrow, a down arrow, a "Next" button, an "Insert" button, and a "Delete" button. At the bottom is a large rectangular area labeled "Menu Layout" with a blue header bar.

The menu editor allows you edit your menus. Each form can have a menu. Menus may be several layers deep.

Caption	The title that appears in the menu
Name	The name of the function that gets called when menu item is selected. The name cannot contain spaces or special characters.
Shortcut	Shortcut key to select item
OK	Save changes to menu and close.
Cancel	Discard changes to menu and close.
Left	Move the highlighted item up a level
Right	Move the highlighted item down a level
Up	Move up to the previous item
Down	move down to the next item
Next	Move to the next item If at the last item, add a new one to the end..

Insert	Add a new item before the selection.
Delete	Delete the selection.

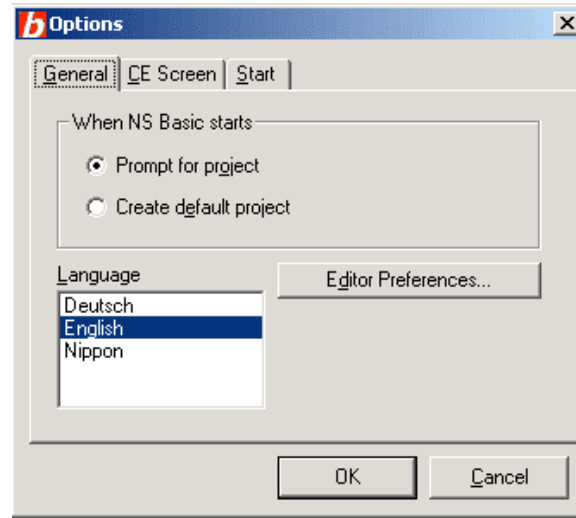
To add code to be executed when a menu item is chosen, close the Menu Editor and select the menu item you want from the CE Screen. A code window will appear with the beginning of the subroutine that will be executed when that menu item is chosen.

The subroutine will be named `menuItem_click`, where `menuItem` is the Name in the Menu Editor . Take care to make sure this name does not conflict with the name of a button, another menu item or other object, even if on another form.

To create a submenu, use the right arrow key to indent the item. Use the left arrow to move a menu item back. The up and down arrows are used to change the order of menu items. Select an item, then use the up and down arrows.

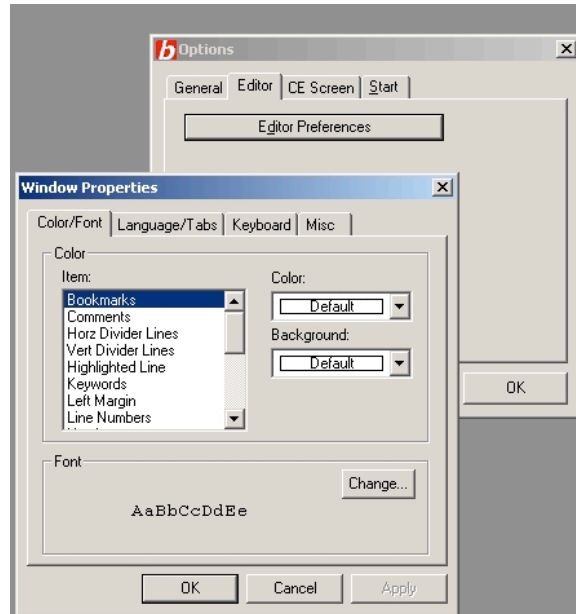
4.9 Options

4.9.1 Options – General



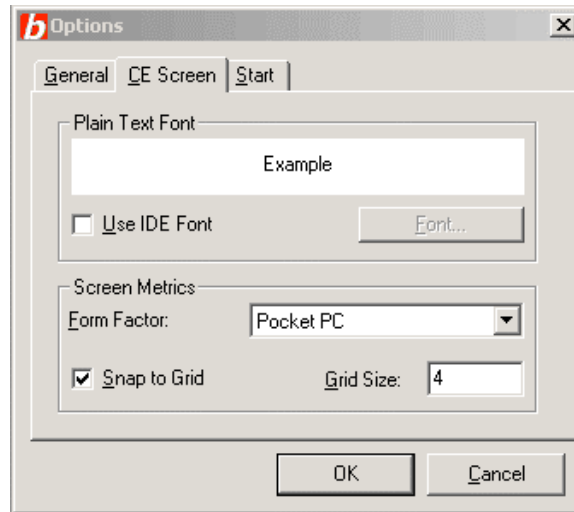
Prompt for project	If selected, at startup NS Basic/CE will ask for if an existing project or new project should be opened.
Create default project	If selected, at startup NS Basic/CE will start with a new project.
Language	Sets the language of messages to be used. Selections will be displayed from the list of files in NS Basic/CE's Lang folder.
Editor Preferences	Set preferences for the Code Window – see 4.9.2.

4.9.2 Options – Editor



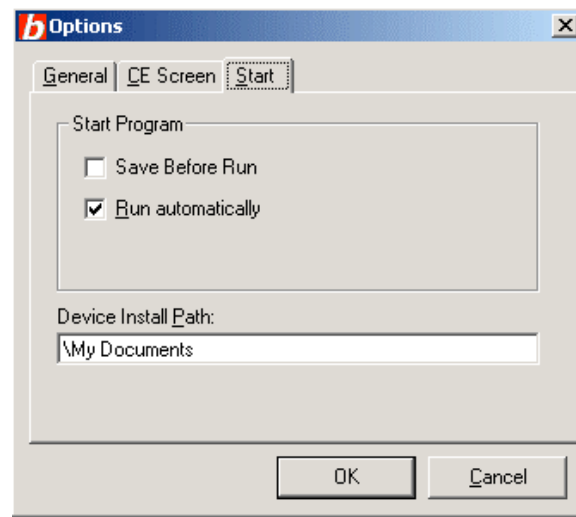
The Editor Option screen allows a wide range of features to be customized for the Code Window, including text coloring, tabs, font, keyboard shortcuts and window appearance.

4.9.3 Options – CE Screen



Snap to Grid	Should objects on the form automatically be aligned to the grid settings? This makes it easier to make nicely laid out forms.
Grid Size	If Snap to Grid is selected, this sets the spacing between grid points.

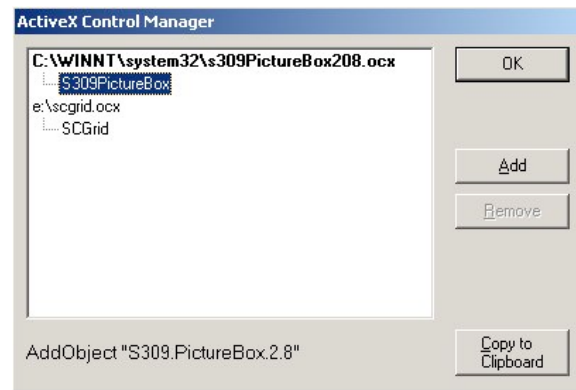
4.9.4 Options – Start



This window sets options that are used when Start on the Run menu.

Save Before Run	Save the file before each Run...Start?
Run automatically	After ActiveSync copies the program, this starts the program running immediately.
Device Install Path	The location on the device to copy the program to. This folder must exist. The correct path will differ depending on the device. For Pocket PC devices, this should be "\\My Documents". For other devices, it should just be "\\".

4.10 ActiveX Control Manager



The ActiveX Control Manager allows you to add controls to the Toolbox. To add a control, click on the Add button to locate the dll or ocx file you wish to add. When one is located, it will be added to the display area with two or more lines if it is a normal ActiveX control. It will be added to the Toolbox when you close the ActiveX Control Manager window.

If you want to add the code to create a control direct into your code, highlight the control you want and click on Copy to Clipboard. A text string containing the first part of the proper ADDOBJECT statement for the control will be generated, displayed on the bottom of the screen and copied to the clipboard so you can paste it to the Code Window.

ActiveX controls that appear in bold lettering are a standard part of NS Basic/Desktop, included in the installation.

4.11 Compiling from the Command Line

You can compile an NS Basic program from the command line, as well as from the Start menu. The format is *nsbasic programName* -compile. The *programName* should be a full path.

Example

```
c:\program files\nsbasic\ce\nsbasic "c:\my
projects\sample.nsb" -compile
```

5. NS Basic/CE Reference

The Reference chapter contains an entry for every Command, Statement, and Function used in NS Basic/CE, in alphabetical order. The entries are listed in the index grouped under Commands, Statements, or Functions.

Each entry in the Reference chapter consists of the following information:

Name	Category
-------------	-----------------

ITEM parameters

Description

This section describes the ITEM and its parameters. Details concerning the uses of ITEM are given, as well as any constraints on its use.

Example

A small program that uses ITEM is listed here.

Output

This section shows the results of running the Example program.

Related Items

A list of zero or more NS Basic/CE Commands, Statements, Functions, or Objects that are related to ITEM. You may often gain a better understanding of ITEM by reviewing the related items.

ABS(*number*)

Description

ABS returns the absolute value, or unsigned magnitude, of a numeric expression. The required parameter, *number*, is any valid numeric expression. If *number* is NULL, ABS returns NULL.

Example

```
REM ABS Example
'ABS returns the absolute value of a number.
PRINT "ABS (-2) = " & ABS (-2)
PRINT "ABS (2) = " & ABS (2)
```

Output

```
ABS (-2) = 2
ABS (2) = 2
```

Related Items

SGN

ADDOBJECT

Statement

ADDOBJECT *objectId:license*[, *objectname*[, *xpos*, *ypos*,
width, *height*, *parent*]]

Description

ADDOBJECT attaches an object to a program. Once an object is attached, its properties can be queried and its methods can be executed from within the program. The required parameter, *objectId*, is a string expression that references the full name of an object from the registry, or the short name of the object. Table 4 below lists some common objects. If the control requires a *license* number, enter it after the *objectId*, separated by a colon. The optional parameter, *objectname*, is a string expression that must follow standard variable naming conventions. *objectname* is the variable that will be added to the program, to refer to the object. If no *objectname* is given *objectId* is used by default. The optional parameters *xpos*, *ypos*, *width*, and *height*, are numeric expressions that specify the default location and size of visual objects in pixels. The *parent* parameter makes the object a child of the object that is named: the new control will be placed on top of the *parent*.

The description of an object and its values are contained in its Properties. (See "Properties")

Objects may contain functions you can call from your program. (See "Methods")

Objects which are contained within NS Basic/CE itself are documented in this Handbook. NS Basic/CE also supports external objects, also known as ActiveX Controls. A number of these come with NS Basic/CE; others are available from third parties. They are documented in the Tech Notes under the Help menu.

Objects can send events to your program by calling SUB procedures. The name of the procedure is a combination of the object name and the event. An optional, comma-separated list of arguments may be included by some events. Important: If an object sends events, you must always supply all 6 arguments. (See "Events")

```
SUB objectname_event(arglist)]  
    'Execute this when objectname gets event  
END SUB
```

Table 4: Common Objects

Short Name	Where it's documented
ADO	Data Base - Tech Note 03
CheckBox	In this document
ComboBox	In this document
Comm	Serial Comms - Tech Note 05
CommandButton	In this document
Date	In this document
Dialog	Dialog boxes - Tech Note 04
File	Simple File I/O - Tech Note 08
FileSystem	Files and Dirs - Tech Note 09
Frame	In this document
Finance	Money calculations - Tech Note 03
Grid	Data Grid - Tech Note 10
HTMLView	Html viewer – Tech Note 42
HScrollbar	In this document
Image	Image holder - Tech Note 03
ImageList	List of Images - Tech Note 11
Label	In this document
ListBox	In this document
ListView	Display Items - Tech Note 12
OptionButton	In this document
Output	In this document
PictureBox	Manipulate Pictures - Tech Note 13
TabStrip	Tab object - Tech Note 17
TextBox	In this document
Time	In this document
TreeView	Tech Note 23
TriButton	In this document
VScrollBar	In this document
WinSock	Internet stuff - Tech Note 03

Example

```
REM ADDOBJECT adds an object to the program
ADDOBJECT "ThirdParty.Control:123-ghs","TPC"
ADDOBJECT "Finance", "Finance"
ADDOBJECT "PictureBox", "PBox", 65, 10, 55,
20
PBox.BorderStyle = 1
PBox.BackColor = vbWhite
PBox.DrawText " Button"
SUB PBox_Click()
    PRINT FORMATCURRENCY(Finance.Pmt(.0058, _
        180, 130000))
END SUB
```

Output**Related Items**

Events, Methods, Properties, SET

AND**Operator**

result = *x* AND *y*

Description

AND returns the logical conjunction of two expressions. *result* is TRUE, if and only if both expressions *x* and *y* evaluate to TRUE, otherwise, *result* is FALSE.

AND also does a bitwise comparison of two numeric expressions. Each bit in *result* is set to 1 if both corresponding bits in *x* and *y* are 1, otherwise it is set to 0.

Example

```
REM AND Example
'AND performs logical and bitwise
conjunction
DIM Test1, Test2, x, y
x = 2
y = 7
Test1 = x > 0 AND y < 10
Test2 = x > 0 AND y > 10
PRINT "Logical:"
PRINT "  x > 0 AND y < 10 = " & Test1
PRINT "  x > 0 AND y > 10 = " & Test2
PRINT "Bitwise:"
PRINT "  x AND y = " & x AND y
```

Output

```
Logical:
  x > 0 AND y < 10 = True
  x > 0 AND y > 10 = False
Bitwise:
  x AND y = 2
```

Related Items

EQV, IMP, NOT, OR, XOR

ARRAY**Function**

ARRAY(*expressionlist*)

Description

ARRAY creates an array dynamically. The required parameter, *expressionlist*, is a comma-delimited list of valid expressions. The resulting array has a length equal to the number of elements in *expressionlist*. Arrays created with ARRAY have a lower bound of 0.

Example

```
REM ARRAY Example
'ARRAY creates an array dynamically
DIM MyArray, Message
'Create an array with 2 string elements
MyArray = Array("Hello", "World!")
'Access the array and concatenate elements
PRINT MyArray(0) & " " & MyArray(1)
```

Output

Hello World!

Related Items

DIM

ASC	Function
-----	----------

ASC(*string*)
ASCB(*string*)
ASCW(*string*)

Description

ASC returns the ANSI character code of a character. The required parameter, *string*, is any valid string expression. If *string* is longer than one character, only the first character is used.

The ASCB function is used with byte data contained in a string. Instead of returning the character code for the first character, ASCB returns the first byte. ASCW is provided for 32-bit platforms that use Unicode characters. It returns the Unicode (wide) character code, thereby avoiding the conversion from Unicode to ANSI.

Example

```
REM ASC Example
'ASC returns an ANSI character code
DIM CapitalA, LowerB
CapitalA = ASC("A is for Apple")
PRINT "Character code for A = " & CapitalA
LowerB = ASC("b")
PRINT "Character code for b = " & LowerB
```

Output

```
Character code for A = 65
Character code for b = 98
```

Related Items

CHR

ATN**Function**

ATN(*number*)

Description

ATN returns the arctangent of a number in radians. The required parameter, *number*, is any numeric expression.

To convert degrees to radians, multiply degrees by $\pi/180$.
To convert radians to degrees, multiply radians by $180/\pi$.

Example

```
REM ATN Example
'ATN calculates the arctangent of a number
DIM Pi
Pi = ATN(1) * 4
PRINT "The value of pi is " & Pi
```

Output

The value of pi is 3.141593

Related Items

COS, SIN, TAN

BREAK	Statement
--------------	------------------

BREAK [*prompt*[, *statements*]]

Description

BREAK temporarily halts execution and opens a dialog box which allows the programmer to execute statements or continue execution. The optional parameter, *prompt*, is a string expression that is displayed in the body of the dialog box. The optional parameter, *statements*, is a string expression that is displayed in the execution area of the dialog box.

Example

```
REM BREAK Example
'BREAK temporarily halts execution
DIM i
FOR i = 0 TO 4
    PRINT (i + 1) * 10
    IF i = 2 THEN
        BREAK "Loop iterator = " & i, "PRINT i"
    END IF
NEXT
```

Output

Related Items

MSGBOX

BYE	Statement
------------	------------------

BYE

Description

BYE terminates the current program and closes the output window. BYE can be called from within a FUNCTION procedure or SUB procedure, but it will have no effect unless the procedure is called in a chain of procedures that was started by an event sent from the operating system.

Example

```
REM BYE Statement
'BYE closes output window and ends the program
DIM When, In10
In10 = MSGBOX("Close in 10 seconds?", vbYESNO)
IF In10 = vbYES THEN
    PRINT "Closing in 10 seconds..."
    When = DATEADD("s", 10, NOW)
    WHILE NOW < When
        'Doing nothing for 10 seconds
    WEND
    BYE
END IF
SUB Output_Click()
    Quit
END SUB
SUB Quit
    BYE
END SUB
```

Output

Open for 10 seconds...

(output window closes)

Related Items

CALL	Statement
-------------	------------------

CALL *procedurename*[(*argList*)]

Description

CALL is an explicit method of executing a FUNCTION procedure or a SUB procedure. The required component, *procedurename*, is any procedure name. The optional component, *argList*, is a comma-delimited list of variables to pass to the called procedure. The CALL keyword is optional, procedures can be executed without the keyword, as *name*[(*argList*)]

Example

```
REM CALL Example
'CALL explicitly executes a procedure
CALL Welcome
CALL Message("NS Basic/CE is excellent.")
Wave
FUNCTION Welcome
    PRINT "Hello World!"
END FUNCTION
FUNCTION Message(Text)
    PRINT "Message: " & Text
END FUNCTION
SUB Wave
    PRINT "Goodbye!"
END SUB
```

Output

```
Hello World!
Message: NS Basic/CE is excellent.
Goodbye!
```

Related Items

SUB, FUNCTION

CHAIN	Statement
-------	-----------

CHAIN *pathname*[, *reset*]

Description

CHAIN loads a program into the environment and begins execution. The required component, *pathname*, is a string expression that specifies the absolute path name of the program to load and execute. The required component, *reset*, is a boolean expression that specifies whether the environment should be completely reset (all constants, variables, and procedures cleared from memory) before loading and executing the new program. Use a BYE statement after CHAIN to complete execution of the calling program.

If *reset* is TRUE, execution continues in the original program until it stops executing. All variable and subroutines are reset and the new program is then loaded and run.

If *reset* is FALSE, then the new program is run immediately, with all variables and subroutines remaining intact. When the new program ends, the original program continues from the next statement. Variables and subroutines added or modified in the CHAINED program will remain.

Hint: Use CHAIN with *reset* FALSE to "include" common program statements in multiple programs.

Example

```
'Program A
PRINT "This is program A"
A=100
CHAIN "B.nsb",TRUE 'reset the environment
PRINT "This is program A after CHAIN"
PRINT A
'Program B
PRINT "This is program B"
PRINT A
A=200
```

Output

```
This is program A
This is program A after CHAIN
100
```

<screen clears>

```
This is program B
```

<no value prints as A is undefined in program B>

Example 2

```
'Program A
PRINT "This is program A"
A=100
```

```
CHAIN "B.nsb",FALSE 'don't reset the
environment
PRINT "This is program A after CHAIN"
PRINT A
'Program B
PRINT "This is program B"
PRINT A
A=200
```

Output

```
This is program A
This is program B
100
This is program A after CHAIN
200
```

Related Items

EXECUTE

CheckBox

Object

ADDOBJECT "CheckBox", *name*, *xpos*, *ypos*, *width*, *height*

ADDOBJECT " TriButton ", *name*, *xpos*, *ypos*, *width*, *height*

Description

CheckBox and TriButton are used to display a button object with a check mark toggle that floats over the output window. TriButton is the same as Check Box, except that it has a third grayed out state. The required component, *name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner. The Value property is 0, when the check mark is not shown; the Value property is 1, when the check mark is shown.

Properties Supported (see "Properties")

Alignment, BackColor, Caption, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Hwnd, Left, Name, ParentHwnd, TabStop, Tag, Text, Timer, Top, Value*, Visible , Width, WindowLong

*The Value property is an Integer (0 or 1). The TriButton has an additional value of 2 for Indeterminate.

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (See "Events")

Click, DblClick, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timer

Example

```
REM CheckBox Example
'CheckBox is a button with a check mark toggle
ADDOBJECT "CheckBox", "Check",10,10,200,20
Check.BackColor = Output.BackColor
Check.Text = "CheckBox Object"
Check.Value = 1
SUB Check_Click
    PRINT "CheckBox Value: " & Check.Value
END SUB
```

Output

☒ CheckBox Object

Example

```
REM TriButton Example
ADDOBJECT "TriButton", "Button", 120, 120,
100, 20
Button.Value = 2 'set to disabled
```

Output

☒ Tri Button

Related Items

ADDOBJECT, Events, Methods, Properties

CHR	Function
-----	----------

CHR(*number*)
CHRB(*number*)
CHRW(*number*)

Description

CHR returns the equivalent character of an ANSI character code. The required expression, *number*, is any numeric expression.

The CHRB function is used with byte data contained in a string. Instead of returning a character, which may be one or two bytes, CHRB always returns a single byte. CHRW is provided for 32-bit platforms that use Unicode characters. Its argument is a Unicode (wide) character code, thereby avoiding the conversion from ANSI to Unicode.

Example

```
REM CHR Example
'CHR returns characters from numbers
DIM Lowercase, Uppercase
Lowercase = CHR(97)
Uppercase = CHR(97 - 32)
PRINT "Lowercase = " & Lowercase
PRINT "Uppercase = " & Uppercase
```

Output

```
Lowercase = a
Uppercase = A
```

Related Items

ASC

CLASS	Statement
-------	-----------

CLASS *name*
[*statements*]
END CLASS

Description

Declares the name of a class, as well as the variables, properties and methods in the class. Variables are defined as PUBLIC or PRIVATE. Properties are defined using PROPERTY SET, PROPERTY LET and PROPERTY GET code blocks. Methods are defined as SUB or FUNCTION blocks. Each class also has optional Class_Initialize and Class_Terminate subroutines that are called when a instance of the class is created or removed. Windows CE 4.0 or later only.

Example

```
Class cGreeter
    Public Sub SayHello(who)
        Print "Hello, " & who & ". Welcome to "
    & Store & "."
    End Sub
    Public Store
    Public Property Get StoreNumber
        Select Case Store
            Case "Main Branch": StoreNumber = 1
            Case "Downtown": StoreNumber = 2
        End Select
    End Property
    Sub Class_initialize
        Store="Main Branch"
    End Sub
    Sub Class_terminate
        Print "Greeter terminated"
    End Sub
End Class

Dim p
Set p = new cGreeter
p.sayHello("Cartman")
Print p.StoreNumber
Set p=Nothing
```

Output

```
Hello Cartman. Welcome to Main Branch.
1
Greeter terminated
```

Related Items

DIM, FUNCTION, PROPERTY, SET, SUB, WITH

ADDOBJECT "ComboBox", *name*, *xpos*, *ypos*, *width*, *height*

Description

ComboBox is used to display a drop-down list object with an entry line that floats over the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and expanded size of the object in pixels, measured from the upper left corner. The Text and Caption properties hold the value of the entry line of the object. The ListIndex property is the index of the entry line in the list, the index of the first item is 0; if the entry line text doesn't match one of the list items, the ListIndex property is -1. Be sure to set the height of the object to its expanded size: at runtime it will be a single line until it is expanded.

Properties Supported (see "Properties")

BackColor, Caption, Enabled, ExpandedHeight (read only), FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, Forecolor, Height, Hwnd, IntegralHeight, Left, list (read only), ListCount (read-only), ListIndex, LowercaseOnly, Name, NewIndex, ParentHWnd, Redraw, Sorted, Style, TabStop, Tag, Text, Timer, Top, TopIndex, UppercaseOnly, Visible, Width, WindowLong

Methods Supported (see "Methods")

AddItem, Clear, Hide, Move, RemoveItem, SetFocus, Show

Events Supported (see "Events")

Change, Click, DropDown, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timer

Example

```
REM ComboBox is a drop-down list object
DIM i
ADDOBJECT "ComboBox", "Combo", 5, 5, 100, 200
Combo.Style = 2
FOR i = vbSUNDAY TO vbSATURDAY
    Combo.AddItem WEEKDAYNAME(i)
NEXT
Combo.ListIndex = WEEKDAY(NOW-1)
PRINT Combo.List(Combo.ListIndex)
```

Output**Related Items**

ADDOBJECT, Events, ListBox, Methods, Properties

CommandButton**Object**

ADDOBJECT "CommandButton", *name*, *xpos*, *ypos*,
width, *height*

Description

CommandButton is used to display a standard button object in the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner of the output window.

Setting the property Value to 1 toggles the button, fires the click event, and sets the value back to 0.

Properties Supported (see "Properties")

BackColor, Caption, Default, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Hwnd, Left, Name, ParentHwnd, TabStop, Tag, Text, Timer, Top, Visible, Width, WindowLong, Value

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Click, DblClick, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timer

Example

```
REM CommandButton Example
'CommandButton is a standard button object
ADDOBJECT "CommandButton", "Button", _
    100, 10, 80, 20
Button.BackColor = Output.BackColor
Button.Text = "Click Me!"
```

Output.

```
BackColor = vbWHITE
SUB Button_Click
    PRINT "Button clicked"
    KillFocus
```

```
END SUB
```

Output**Related Items**

ADDOBJECT, Events, Methods, Properties

CONST	Statement
-------	-----------

[PUBLIC | PRIVATE] CONST *name* = *expression*

Description

CONST declares constants, which can be used in expressions, in place of literal values. The required component, *name*, must follow standard variable naming conventions. The required component, *expression*, is any literal, constant, or combination that includes all arithmetic or logical operators (except IS).

The optional PUBLIC and PRIVATE keywords are used at script level, to designate constants as visible or invisible to FUNCTION procedures and SUB procedures. By default, all constants are PUBLIC, and available throughout the script and in all procedures. All constants declared inside procedures are available only within the procedure.

Multiple constants may be declared on a single line, by separating each constant assignment with a comma. If a PUBLIC or PRIVATE keyword is used in this manner, it will be applied to all constants declared on that line.

Example

```
REM CONST Example
'CONST defines constants
CONST SHAPE = "Rectangle"
PRIVATE CONST AREA = 51
PUBLIC CONST LENGTH = 7, WIDTH = 11
PrintArea LENGTH, WIDTH
SUB PrintArea(l, w)
    DIM Area
    Area = l * w
    PRINT SHAPE & " area: " & l & " * " & w & _
    " = " & Area
END SUB
```

Output

Rectangle area: 7 * 11 = 77

Related Items

FUNCTION, PUBLIC, PRIVATE, SUB

Conversions	Function
-------------	----------

CBOOL(*expression*)
 CBYTE(*expression*)
 CCUR(*expression*)
 CDATE(*expression*)
 CDBL(*expression*)
 CINT(*expression*)
 CLNG(*expression*)
 CSNG(*expression*)
 CSTR(*expression*)

Description

The conversion functions return an expression that has been converted to the appropriate type. The required parameter, *expression*, is any valid expression.

If the return value lies outside the acceptable range of its return type, an error occurs.

Table 5: Conversion Functions

Function	Returns	Comments
CBOOL	Boolean	False if expression is zero True otherwise
CBYTE	Byte	A whole number that ranges from 0 to 255
CCUR	Currency	A currency datum
CDATE	Date	Date range January 1, 100 to December 31, 9999, Valid expressions are date expressions, or date/time literals
CDBL	Double	Number ranges from -1.79769313486232E308 to -4.94065645841247E-324 for negative values, and from 4.94065645841247E-324 to 1.79769313486232E308 for positive values
CINT	Integer	Number ranges from -32,768 to 32,767, values with fractional parts (<i>fp</i>) are rounded <i>fp</i> < 0.5 rounded down <i>fp</i> > 0.5 rounded up <i>fp</i> = 0.5 rounded to nearest even number
CLNG	Long Integer	Number ranges from -2,147,483,648 to

		2,147,483,647, values with fractional parts (<i>fp</i>) are rounded $fp < 0.5$ rounded down $fp > 0.5$ rounded up $fp = 0.5$ rounded to nearest even number
CSNG	Single	Number ranges from -3.403823E38 to -1.401298E-45 for negative numbers, and from 1.401298E-45 to 3.403823E38 for positive numbers
CSTR	String	<ul style="list-style-type: none"> Booleans converted to "True" or "False" Dates converted to short-date format of system Errors converted to "Error <number>" Numbers converted to string containing the number

Example

```
REM Conversion Functions Example
PRINT "CBYTE(99.44) = " & CBYTE(99.44)
PRINT "CCUR(9283.066) = " & CCUR(9283.066)
PRINT "CDATE(8/18/98) = " & CDATE("8/18/98")
PRINT "CDBL(3.141593) = " & CDBL("3.141593")
PRINT "CINT(3.141593) = " & CINT("3.141593")
PRINT "CSNG(10) = " & CSNG(10)
PRINT "CSTR(TRUE) = " & CSTR(TRUE)
```

Output

```
CBYTE(99.44) = 99
CCUR(9283.066) = 9283.066
CDATE(8/18/98) = 8/18/1998
CDBL(3.141593) = 3.141593
CINT(3.141593) = 3
CSNG(10) = 10
CSTR(TRUE) = True
```

Related Items

Is Functions

COS**Function**

`COS(number)`

Description

COS calculates the cosine of a number expressing an angle in radians. The required parameter, *number*, is any numeric expression. The value returned is a double-precision floating-point number that can range from -1 to 1.

To convert degrees to radians, multiply degrees by $\pi/180$.
To convert radians to degrees, multiply radians by $180/\pi$.

Example

```
REM COS Example
'COS calculates the cosine of a number
PRINT "The cosine of 0 is " & COS(0)
```

Output

The cosine of 0 is 1

Related Items

SIN, TAN

CREATESHORTCUT	Function
-----------------------	-----------------

CreateShortCut(*ShortcutName*, *TargetPath*)

Description

Creates a directory shortcut to an executable. Returns true if successful, false if it cannot create the shortcut or it already exists. The target path should include quote marks, so it can be parsed properly..

Example

REM CreateShortcut Example

'This will create a link in the Programs
folder to your program.

```
err = CreateShortcut("\Windows\Start  
Menu\Programs\My Test.exe", "\"" &  
currentPath & "\"")
```

Output

Related Items

CURRENTPATH**Global**

CurrentPath

Description

CurrentPath returns a string the with the path to the current program. If the program has not been saved yet, it is empty. This global property can be read, but not set.

Example

```
REM CurrentPath Example
PRINT "The path to the current program is "
& CurrentPath
```

Output

The path to the current program is \test.nsb

Related Items

GETCOMMANDLINE

DATE	Function
------	----------

DATE

Description

DATE returns the current system date.

Example

```
REM DATE Example
'DATE returns current system date
DIM Today
Today = DATE
PRINT "Today is " & Today
```

Output

Today is 8/18/1998

(sample date output is system dependant)

Related Items

NOW, TIME

Date**Object**

ADDOBJECT "Date", *name*, *xpos*, *ypos*, *width*, *height*

Description

Date is used to display a standard date pickerobject in the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner of the output window. Do not use a MSGBOX inside your Date_Change event: it will cause an error. This object is not available on Windows CE 2.0 devices. Use the .Date property to set the date.

Properties Supported (see "Properties")

BorderStyle, Date, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, LongFormat, Height, Hwnd, Left, Name, ParentHwnd, TabStop, Tag, Text, Timer, Top, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Change, DropDown, gotFocus, KeyDown, KeyUp, KeyPress, LostFocus, Timer

Example

```
REM Date Example
ADDOBJECT "Date", "Date", 100,100,90,20
Date.fontBold=true
SUB Date_Dropdown
    PRINT "Date Dropdown"
END SUB
SUB Date_Change
    PRINT "Date Changed to " & Date.Date
END SUB
```

Output**Related Items**

ADDOBJECT, Events, Methods, Properties

DATEADD**Function**

DATEADD(*interval*, *number*, *date*)

Description

DATEADD returns the date which is obtained by adding a specified number of date intervals to a given date. The required parameter, *interval*, is a string expression that specifies the type of date interval to add, see Table 6 below. The required parameter, *number*, is any numeric expression that specifies the number of intervals to add. The required parameter, *date*, can be any expression that represents a date.

Table 6: Interval values

Value	Description
yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute
s	Second

Example

```
REM DATEADD Example
'DATEADD adds date intervals to a date
PRINT "+10 seconds:", DATEADD("s", 10, NOW)
PRINT "-1 year:", DATEADD("yyyy", -1, NOW)
```

Output

```
+10 seconds:  8/18/98 10:52:54 PM
-1 year:      8/18/97 10:52:44 PM
```

Related Items

DATEDIFF, DATEPART

DATEDIFF Function

DATEDIFF(*interval*, *date1*, *date2*[, *firstdayofweek*[, *firstweekofyear*]])

Description

DATEDIFF returns the number of intervals between two dates. The required parameter, *interval*, is a string expression, see Table 6.. The required parameters, *date1* and *date2*, can be any expressions that represent dates. The optional parameter *firstdayofweek* is Sunday, if not specified. The optional *firstweekofyear* is the week containing January 1, if not specified.

Table 7: firstdayofweek constants

Constant	Value	Description
vbUseSystem	0	NLS API setting
vbSunday	1	Sunday (default)
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Table 8: firstweekofyear constants

Constant	Value	Description
vbUseSystem	0	NLS API setting
vbFirstJan1	1	Week with January 1
vbFirstFourDays	2	First week of year with at least 4 days
vbFirstFullWeek	3	First full week of year

Example

```
REM DATEDIFF Example
'DATEDIFF calc difference between 2 dates
DIM Born
Born = INPUTBOX("Enter your birthdate")
Born = CDATE(Born)
PRINT "Since " & Born & " there have been"
PRINT DATEDIFF("d", Born, NOW) & " days"
PRINT "or"
PRINT DATEDIFF("n", Born, NOW) & " minutes"
```

Output

Since 12/27/1970 there have been
10096 days
or
14539612 minutes

(sample date output is system dependant)

Related Items

DATEADD, DATEPART

DATEPART	Function
----------	----------

DATEPART(*interval*, *date*[, *firstdayofweek*[,
firstweekofyear]])

Description

DATEPART returns a number specifying a part of the given date. The required parameter, *interval*, determines the part of date which is measured and returned. The optional parameter *firstdayofweek* is Sunday, if not specified. The optional *firstweekofyear* is the week containing January 1, if not specified.

Example

```
REM DATEPART Example
'DATEPART returns a number from part of a date
DIM QuarterPart, MonthPart, DayPart
QuarterPart = DATEPART("q", NOW)
MonthPart = DATEPART("m", NOW)
DayPart = DATEPART("d", NOW)
PRINT "Today is day " & DayPart
PRINT "of month " & MonthPart
PRINT "in quarter " & QuarterPart
```

Output

```
Today is day 18
of month 8
in quarter 3
(sample output from August 18, 1998)
```

Related Items

DATEADD, DATEDIFF

DATESERIAL**Function**

DATESERIAL(*year, month, day*)

Description

DATESERIAL returns a date constructed from a given year, month, and day. The required parameter, *year*, is any numeric expression ranging from 100 to 9999. The required parameters, *month* and *day*, can be any numeric expression.

Example

```
REM DATESERIAL Example
'DATESERIAL builds a date from its parts
DIM IndepDay, Birthday
IndepDay = DATESERIAL(1776, 7, 4)
Birthday = DATESERIAL(1970, 12, 27)
PRINT "Independence Day:", IndepDay
PRINT "My birthday:", Birthday
```

Output

```
Independence Day:      7/4/1776
My birthday:    12/27/1970
```

(sample date output is machine dependant)

Related Items

TIMESERIAL

DATEVALUE**Function**

DATEVALUE(*date*)

Description

DATEVALUE returns a date. The required parameter, *date*, is usually a string, but any expression that can represent a date, a time, or a date and time ranging from January 1, 100 to December 31, 9999, can be used.

If *date* is a string that consists of numbers separated by date separators, it recognizes the order for month, day, and year according to the Short Date format specified for your system. If the year is left out of *date*, the current year is used.

Unambiguous month names as strings will be recognized in either long or abbreviated form.

Date separators are characters that separate day, month, and year when a date is formatted as a string, they are determined by your system settings.

Example

```
REM DATEVALUE Example
'DATEVALUE returns a date
DIM IndepDay, Birthday
IndepDay = DATEVALUE("July 4, 1776")
Birthday = DATEVALUE("Dec 27 1970")
PRINT "Independence Day:", IndepDay
PRINT "My birthday:", Birthday
```

Output

```
Independence Day:      7/4/1776
My birthday:    12/27/1970
```

(sample date output is machine dependant)

Related Items

FORMAT, TIMEVALUE

DAY**Function**

DAY(*date*)

Description

DAY returns an integer, ranging from 1 to 31, that represents the day of the month, from a given date. The required parameter, *date*, can be any expression that represents a date.

Example

```
REM DAY Example
'DAY returns number representing day of month
DIM Today, NextQuarter
Today = DATE
NextQuarter = Today + 90
PRINT "Today's day is " & DAY(Today)
PRINT "90 days from now it will be " _
      & DAY(NextQuarter)
```

Output

```
Today's day is 18
90 days from now it will be 16
```

Related Items

HOUR, MINUTE, MONTH, NOW, SECOND, WEEKDAY, YEAR

DECLARE	Statement
----------------	------------------

```
DECLARE "Sub name Lib ""libname"" [Alias  
""aliasname"" ]([arglist])"  
Declare "Function name Lib ""libname"" [Alias  
""aliasname"" ]([arglist]) [As type]"
```

Description

Declare an API function for use. The *name* is the name of the function to use as a function or subroutine in an NS Basic program. When this internal name within the DLL is different, use the Alias clause giving the internal name of the function. This is useful when an internal name conflicts with a keyword. The *libname* is the name of the DLL library. It may be a full pathname to the DLL or a partial pathname or single file name, in which case the system searches for the DLL. Most normal DLLs will be stored in the Windows directory, in which case the path name can be omitted, and the system will find it quickly. See the Tech Note on Declare for additional documentation.

Example

```
Rem Get storage info on device  
Dim lpAvailable, lpTotalBytes, lpTotalFree  
Declare "Function GetDiskFreeSpaceExW Lib  
""Coredll"" (_  
    ByVal lpDirectoryName As String, _  
    ByRef lpFreeBytesAvailable As Long, _  
    ByRef lpTotalNumberOfBytes As Long, _  
    ByRef lpTotalNumberOfFreeBytes As Long) _  
    As Boolean"  
res = GetDiskFreeSpaceExW("e:", _  
    lpAvailable, lpTotalBytes, lpTotalFree)  
MsgBox "Available: " & lpAvailable
```

Output

Available: 123,000,004

Related Items

DIM	Statement
DIM <i>nameA</i> [[<i>subscriptA</i> [, <i>subscriptB</i> [, <i>subscriptC</i> ...]]]] [, <i>nameB</i> ...[, <i>nameC</i> ..., [...]]]	

Description

DIM is used to declare variables and allocate storage space. The required component, *nameA*, is the name of the variable, it must follow standard variable naming conventions. The optional list of *subscripts* are the upper bounds of dimensions of an array variable. Up to 60 comma-separated dimensions may be declared. Script level variables are available to all procedures in the script; procedure level variables are available only in the procedure they are declared in. The lower bound of an array is always zero.

Example

```
REM DIM Example
'DIM declares variables and allocates storage
'An empty variable named "Foo"
DIM Foo
'A one-dimensional array with 10 elements
DIM OneD(9)
'A two-dimensional array with 600 elements
'20 x 30
DIM TwoD(19, 29)
```

Related Items

ARRAY, REDIM, SET

DOEVENTS	Statement
-----------------	------------------

DOEVENTS

Description

DOEVENTS is used to allow events to be processed during long, intensive loops. When DOEVENTS is called, any screen or Timer events will be processed, and then the loop will continue.

Example

```
REM DOEVENTS Example
'DOEVENTS allows screen events to occur
'during long loops
FOR i = 0 TO 100000
    j = j + 1
    DOEVENTS
NEXT
```

Output

(nothing here – but other events can happen)

Related Items

DO...LOOP	Statement
-----------	-----------

```
DO [WHILE | UNTIL condition]  
  [statements]  
[EXIT DO]  
  [statements]  
LOOP
```

or

```
DO  
  [statements]  
[EXIT DO]  
  [statements]  
LOOP [WHILE | UNTIL condition]
```

Description

DO...LOOP repeats a block of *statements* WHILE a given condition is TRUE, or UNTIL a given condition becomes TRUE. The required component, *condition*, is any expression that can be evaluated as TRUE or FALSE. The optional component, *statements*, are the statements executed during the body of the loop. Any number of optional EXIT DO statements can be used to exit a loop before it is finished. DO...LOOP statements can be nested, and any EXIT DO statements in a nested loop transfer execution to one level above the loop where the EXIT DO occurs.

Placing the WHILE/UNTIL clause directly after DO causes *condition* to be evaluated before the first loop is executed, if *condition* is FALSE *statements* are never executed; placing the WHILE/UNTIL clause after LOOP causes the body of the loop to be executed before condition is evaluated, *statements* will always be executed, at least once.

Example

```
REM DO...LOOP Example  
'DO...LOOP repeats a block of statements  
DIM Counter  
Counter = 1  
'Loop that prints 1 to 5  
DO WHILE Counter < 6  
  PRINT "DO WHILE...LOOP:", Counter  
  Counter = Counter + 1  
LOOP  
PRINT  
'Infinite loop that uses EXIT DO to terminate  
Counter = 1000  
DO  
  PRINT "Infinite Loop:", Counter  
  IF Counter >= 1000000 THEN  
    EXIT DO
```

```
Counter = Counter * 10  
LOOP
```

Output

```
DO WHILE...LOOP:      1  
DO WHILE...LOOP:      2  
DO WHILE...LOOP:      3  
DO WHILE...LOOP:      4  
DO WHILE...LOOP:      5  
Infinite Loop: 1000  
Infinite Loop: 10000  
Infinite Loop: 100000  
Infinite Loop: 1000000
```

Related Items

EXIT, FOR EACH...NEXT, FOR...NEXT, WHILE...WEND

result = *x* EQV *y*

Description

EQV returns the logical equivalence of two expressions. *result* is TRUE, if both expressions *x* and *y* evaluate to TRUE or both expressions *x* and *y* evaluate to FALSE, otherwise, *result* is FALSE.

EQV also does a bitwise comparison of two numeric expressions. Each bit in *result* is set to 1 if both corresponding bits in *x* and *y* are 1 or both corresponding bits in *x* and *y* are 0, otherwise it is set to 0.

Example

```
REM EQV Example
'EQV performs logical and bitwise equivalence
DIM Test1, Test2, Test3, x, y
x = 4
y = 9
Test1 = x < 0 EQV y < 10
Test2 = x > 0 EQV y > 10
Test3 = x < 0 EQV y > 10
PRINT "Logical:"
PRINT "  x < 0 EQV y < 10 = " & Test1
PRINT "  x > 0 EQV y > 10 = " & Test2
PRINT "  x < 0 EQV y > 10 = " & Test3
PRINT "Bitwise:"
PRINT "  x EQV y = " & (x EQV y)
```

Output

```
Logical:
  x < 0 EQV y < 10 = False
  x > 0 EQV y > 10 = False
  x < 0 EQV y > 10 = True
Bitwise:
  x EQV y = -14
```

Related Items

AND, IMP, NOT, OR, XOR

ERASE**Statement**

ERASE *arrays*

Description

ERASE reinitializes fixed-size arrays, and releases memory allocated for dynamic-array storage. The required component, *arrays*, is a comma separated list of one or more array variables.

Example

```
REM ERASE Example
'ERASE reinitializes arrays
DIM Children(3)
Children(0) = "Eric"
Children(1) = "Kenny"
Children(2) = "Kyle"
Children(3) = "Stan"
PrintArray Children, 4
ERASE Children
PrintArray Children, 4
FUNCTION PrintArray(arr, elements)
    DIM i
    FOR i = 1 to elements
        PRINT "#" & i & ":", "(" & arr(i - 1) & ")"
    NEXT
    PRINT
END FUNCTION
```

Output

```
#1:      (Eric)
#2:      (Kenny)
#3:      (Kyle)
#4:      (Stan)
#1:      ()
#2:      ()
#3:      ()
#4:      ()
```

Related Items

DIM, ARRAY

ERR**Object**

Err
Err.Number
Err.Description
Err.Source
Err.Clear
Err.Raise

Description

The Err object is used to manage run-time errors. The Err object is always available while a program is running. There is no need to use ADDOBJECT. When an error occurs, the Number, Description, and Source properties are set in the Err object. Once an error has been handled, the Clear method resets the Err object, removing any error data from its properties. The Raise method can be used to generate an error event.

Example

```
REM Err Object Example
'Err object manages run-time errors
DO_DIV0(3)
SUB DO_DIV0(Num)
    ON ERROR RESUME NEXT
    PRINT Num / 0
    IF Err THEN
        PRINT Err.Number, Err.Description
        Err.Clear
    END IF
END SUB
```

Output

```
11      Division by zero
```

Related Items

ON ERROR

ESCAPE	Function
--------	----------

ESCAPE(*string*)
UNESCAPE(*string*)

Description

ESCAPE returns a string with all special characters turned into a % character followed by the hex value of the character

UNESCAPE converts a string with special characters turned into a normal string.

Example

```
PRINT ESCAPE ("ABC!@#$$%")
PRINT UNESCAPE ("ABC%0d%0aDEF")
```

Output

```
ABC%21@%23%24%25
ABC
DEF
```

Related Items

UNESCAPE

EVAL**Function**

EVAL(*string*)

Description

EVAL returns a value created by executing an expression as if it were a FUNCTION procedure. The required parameter, *string*, is a string expression that is executed. If multiple statements are to be executed, separate them with a carriage return (vbCRLF). The temporary, virtual procedure that gets created has all program variables passed in by value, so the variables are unmodifiable by the EVAL function.

Example

```
REM EVAL Example
'EVAL execute a string as a FUNCTION
DIM x
x = 5
PRINT EVAL("x")
EVAL("x = x * 10")
PRINT x
```

Output

```
5
5
```

Related Items

EXECUTE

Events**Object**

```
[PUBLIC] SUB ObjectName_Event[(arglist)]  
  [Statements]  
END SUB
```

Description

Object events are triggered either programmatically or through user interaction. When an event is triggered, an object calls a SUB procedure in the program. The name of the procedure, *ObjectName_Event*, is a combination of the object name and the name of event. The optional component, *arglist*, is a comma-separated list of arguments that may be included in the procedure call by some events.

Table 9: Object Events

Event	Comments
Change	ComboBox (item selected or text input), ListBox (item selected), TextBox (Text changed)
Click	
DbClick	ListBox, TextBox
DropDown	ComboBox, Date
GotFocus	Object activated to receive keyboard input.
Form_Hide	Generated code – do not modify
Form_Load	Called from Form_Show
Form_Show	Generated code – do not modify
Form_Unload	Called from Form_Hide
KeyDown	Keycode, shift as args Shift=1, 2-CTRL, 4=Alt
KeyPress	Char as arg
KeyUp	Keycode, shift as args Shift=1, 2-CTRL, 4=Alt
LostFocus	Object deactivated
Output_Close	Called when app is closed
Output_Size	Called when output size is changed or screen rotated
Timer	CheckBox, ComboBox, CommandButton, Frame, HScrollBar, Label, ListBox, OptionButton, TextBox, VScrollBar

Note: If a program does not have a procedure to respond to an event when it is called, no error will occur.

Example

```
REM Object Events Example
'Object Events call procedures in the program
DIM When
ADDOBJECT "ComboBox", "Combo", 5, 30, 150, 80
Combo.Style = 2
SUB Combo_DropDown
    Combo.Clear
    Combo.AddItem DATE
    Combo.AddItem DATEADD("d", 1, DATE)
    Combo.AddItem DATEADD("ww", 1, DATE)
END SUB
SUB Combo_Click
    When = Combo.Text
    PRINT "Item selected: " & When
END SUB
```

Output

Related Items

Methods, Properties

EXECUTE	Statement
----------------	------------------

```
EXECUTE(string)  
EXECUTE("file:|ascii:|unicode:" & string)  
NSEXECUTE("file:|ascii:|unicode:" & string)
```

Description

EXECUTE executes an expression or file as if it were code substituted in the program. The required parameter, *string*, can be either a string expression or the name of a file containing statements that is executed. If multiple statements are to be executed, separate them with a carriage return (vbCRLF). The code can access and modify all variables in the current running program.

If the string starts with "file:" or "ascii:", the rest of the string is used as a pathname to an ascii file. If the string starts with "unicode:" it refers to a file in unicode format. The filename feature is only available at development NS Basic/CE - not the Runtime version.

NSEXECUTE works the same way as EXECUTE, but objects created using ADDOBJECT will send events back to your program.

Example

```
REM EXECUTE execute a string as a SUB  
DIM x  
x = 5  
EXECUTE("PRINT x * 10")  
EXECUTE("x = x * 10")  
'text.txt is a file with "PRINT 50" in it  
EXECUTE("ascii:\my documents-est.txt")  
PRINT x
```

Output

```
50  
50  
50110
```

Related Items

EVAL, EXECUTEGLOBAL

EXECUTEGLOBAL	Statement
---------------	-----------

EXECUTEGLOBAL(*string*)

Description

EXECUTEGLOBAL executes one or more statements in the global namespace of a script. The required parameter, *string*, can be either a string expression or the name of a file containing statements that is executed. If multiple statements are to be executed, separate them with a carriage return (vbCRLF) or colon. All statements used with EXECUTEGLOBAL are executed in the program's global namespace. This allows code to be added to the program so that any procedure can access it. For example, a CLASS statement can be executed at run time and functions can subsequently create new instances of the class.

Adding procedures and classes at runtime can be useful, but also introduces the possibility of overwriting existing global variables and functions at runtime. If you don't need access to a variable or function outside of a procedure, use the EXECUTE statement that will only affect the calling function.

Example

```
REM EXECUTEGLOBAL Example
'EXECUTEGLOBAL executes a string as a SUB
DIM x
x = ("DIM y" & vbCRLF & "y = 4 * 10" & _
    vbCRLF & "Print y")
EXECUTEGLOBAL x
```

Output

40

Related Items

EVAL, EXECUTE

EXIT	Statement
-------------	------------------

EXIT DO
EXIT FOR
EXIT FUNCTION
EXIT SUB

Description

EXIT terminates the execution of a block of code in a DO...LOOP, FOR...NEXT, FOR EACH...NEXT, FUNCTION, or SUB. When used to exit a loop, either DO...LOOP, FOR...NEXT, or FOR EACH...NEXT, execution continues with the first statement after the loop; if the loop is nested inside another loop, control is transferred to the loop outside the loop where the EXIT is encountered. When used to exit a FUNCTION or SUB procedure, execution continues with the first statement after the statement which called the procedure.

Example

```
REM EXIT Example
'EXIT terminates loops and procedures
DIM i
FOR i = 1 to 10
    IF i > 1 THEN
        EXIT FOR
    END IF
    PRINT "Attempting to do nothing"
    DoNothing
NEXT
PRINT "Done"
SUB DoNothing
    EXIT SUB
    PRINT "This statement is never executed"
END SUB
```

Output

Attempting to do nothing
Done

Related Items

FOR...NEXT, DO...LOOP, FUNCTION, PROPERTY, SUB

EXP**Function**

EXP (*number*)

Description

EXP returns a double-precision value equal to e^{number} . The required parameter, *number*, is any numeric expression. e is the base of natural logarithms, and is approximately equal to 2.718282.

Example

```
REM EXP Example
'EXP raises a number to the eth power
PRINT "EXP(0) = " & EXP(0)
PRINT "e = " & EXP(1)
```

Output

```
EXP(0) = 1
e = 2.718282
```

Related Items

LOG

FILTER**Function**

FILTER (*stringarray*, *value*[, *include*[, *compare*]])

Description

FILTER returns an array of strings that is a subset of inputs that have met the specified filter criteria. The required parameter, *stringarray*, is a one-dimensional array of strings to be filtered. The required parameter, *value*, is a string expression to be searched for. The optional parameter, *include*, is a boolean value, when TRUE, the returned values are the strings that contain *value*, when FALSE, the returned values do not contain *value*. The default value of *include* is TRUE. The optional parameter, *compare*, is a numeric expression or constant that specifies the type of comparisons to perform, see below. The default value of *compare* is vbBinaryCompare.

Table 10: Comparison constants

Constant	Value	Description
vbBinaryCompare	0	Binary comparison, case sensitive (default)
vbTextCompare	1	Textual comparison, case insensitive

Example

```
REM FILTER Example
'FILTER finds matches in an array of strings
DIM Who, TheKs, NotEric
Who = ARRAY("Eric", "Kenny", "Kyle", "Stan")
TheKs = FILTER(Who, "k", TRUE, vbTextCompare)
NotEric = FILTER(Who, "Eric", FALSE, _
    vbBinaryCompare)
PrintArray "Who", Who
PrintArray "The K's", TheKs
PrintArray "Everyone but Eric", NotEric
SUB PrintArray(ArrName, Arr)
    DIM i
    PRINT ArrName
    FOR i = 0 TO UBOUND(Arr)
        PRINT "  " & Arr(i)
    NEXT
END SUB
```

Output

```
Who
  Eric
  Kenny
  Kyle
```

Stan
The K's
Kenny
Kyle
Everyone but Eric
Kenny
Kyle
Stan

Related Items

REPLACE

`FIX(number)`

Description

FIX removes the fractional part from a number, returning the integer closest to 0. The required parameter, *number*, is any numeric expression. When *number* is positive, the next smallest integer is returned; when *number* is negative, the next largest integer is returned. The required parameter, *number*, is any valid numeric expression.

Example

```
REM FIX Example
'FIX converts floats to int nearest 0
DIM Pos, Neg
Pos = EXP(1)
Neg = -EXP(1)
PRINT "FIX(e) = " & FIX(Pos)
PRINT "FIX(-e) = " & FIX(Neg)
```

Output

```
FIX(e) = 2
FIX(-e) = -2
```

Related Items

INT

FOR...NEXT	Statement
-------------------	------------------

```
FOR counter = start TO end [STEP step]  
    [statements]  
    [EXIT FOR]  
    [statements]  
NEXT
```

Description

FOR...NEXT repeats a group of statements. The required component, *counter*, is a number that can be used to reference the current iteration. The required components, *start* and *end*, are the initial and final values of *counter*, each can be specified with any valid numeric expression. The optional parameter, *step*, can be used to set the amount counter is incremented each loop, the default is 1. The optional component, *statements*, will be executed as the body of the loop. Any number of optional EXIT FOR statements can be used to exit a loop before it is finished. FOR...NEXT statements can be nested, and any EXIT FOR statements in a nested loop transfer execution to one level above the loop where the EXIT FOR occurs.

Example

```
REM FOR...NEXT Example  
'FOR...NEXT repeats a group of statements  
DIM Puppets  
Puppets = ARRAY("Hat", "Twig")  
FOR i = 0 to 1  
    PRINT "Puppet: Mr. " & Puppets(i)  
NEXT  
FOR i = 0 to 10 STEP 5  
    PRINT i  
NEXT
```

Output

```
Puppet: Mr. Hat  
Puppet: Mr. Twig  
0  
5  
10
```

Related Items

DO...LOOP, EXIT, FOR EACH...NEXT, WHILE...WEND

FOR EACH...NEXT	Statement
------------------------	------------------

```
FOR EACH element IN group
    [statements]
    [EXIT FOR]
    [statements]
NEXT [element]
```

Description

FOR EACH...NEXT repeats a group of statements, once for each element in an array or collection. The required parameter, *element*, is a variable name that can be used to reference the current element. The required parameter, *group*, is the name of an array, or collection of objects. The optional component, *statements*, will be executed as the body of the loop. Any number of optional EXIT FOR statements can be used to exit a loop before it is finished. FOR EACH...NEXT statements can be nested, and any EXIT FOR statements in a nested loop transfer execution to one level above the loop where the EXIT FOR occurs.

Note: FOR EACH...NEXT will not work with arrays of user-defined types.

Example

```
REM FOR EACH...NEXT Example
DIM School
School = ARRAY("Principal", "Mr. Garrison", _
    "Chef")
FOR EACH Employee IN School
    PRINT "School employee:", Employee
NEXT
```

Output

```
School employee:      Principal
School employee:      Mr. Garrison
School employee:      Chef
```

Related Items

DO...LOOP, EXIT, FOR...NEXT, WHILE...WEND

Format	Functions
---------------	------------------

```

FORMATCURRENCY(expression[, fractionaldigits[,
    leadingdigit[, parensfornegative[, groupdigits]]]])
FORMATDATETIME(date[, formatname])
FORMATNUMBER(expression[, fractionaldigits[,
    leadingdigit[, parensfornegative[, groupdigits]]]])
FORMATPERCENT(expression[, fractionaldigits[,
    leadingdigit[, parensfornegative[, groupdigits]]]])
    
```

Description

FORMATCURRENCY, FORMATNUMBER, and FORMATPERCENT return a string obtained by formatting the given expression as a currency, number, or percent. The required parameter, *expression*, is any valid expression of the given type. The optional parameter, *fractionaldigits*, is a numeric expression representing the number of digits to print after the decimal, the default is -1 which uses the system settings. The optional parameters, *leadingdigit*, *parensfornegative*, and *groupdigits*, are numeric expressions or constants, see below. *leadingdigit* specifies whether a leading zero is printed for numbers with only fractional parts. *parensfornegative*, specifies if negative values are to be printed with enclosing parentheses. *groupdigits* specifies if numbers are to be printed in groups using the systems group delimiter

Table 11: Tristate values

Name	Value	Description
True	-1	True
False	0	False
UseDefault	-2	Use system settings

FORMATDATETIME returns a string obtained by formatting a date. The required parameter, *date*, is any valid expression that represents a date. The optional parameter, *formatname*, is a numeric expression or constant that specifies how the date is formatted.

Table 12: formatname constants

Constant	Value	Description
vbGeneralDate	0	Short date, Long time
vbLongDate	1	Long date
vbShortDate	2	Short date
vbLongTime	3	Long time
vbShortTime	4	Short time

Example

```
REM Format Functions
DIM UseDefault
UseDefault=-2
'Currency
PRINT FORMATCURRENCY(-3.5, -1, _
    TristateUseDefault, True)
PRINT FORMATCURRENCY(123456, 0, True, False,
True)
'Date/Time
PRINT FORMATDATETIME(NOW)
PRINT FORMATDATETIME(Birthdate, vbLongDate)
'General Numbers
PRINT FORMATNUMBER(-0.1429, 6, False)
PRINT FORMATNUMBER(987654.321, 3, True,
False, True)
'Percentages
PRINT FORMATPERCENT(0.007, 2, False)
PRINT FORMATPERCENT(1234.56, 0, True, False,
False)
```

Output

```
($3.50)
$123,456
8/18/1998 10:44 PM
August 18, 1998
-.142900
987,654.321
.70%
123456%
```

(sample output is system dependant)

Related Items

FRAME

Object

ADDOBJECT "Frame", *name*, *xpos*, *ypos*, *width*, *height*, *parent*

Description

A frame is a container that can contain other objects. To set an object as a child of another object, set *parent* to the name of the parent object. Methods applied to the frame will then automatically be applied to its children. The bounds of a child are relative to the frame it is contained in.

Properties Supported (see "Properties")

BackColor, Caption, Enabled, Font, FontBold, FontItalic, FontStrikeThru, FontUnderline, FontName, FontSize, ForeColor, Height, HWnd, Left, Name, ParentHWnd, Tag, Top, Timer, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, Show

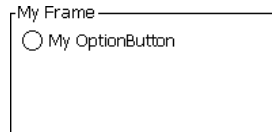
Events Supported (see "Events")

Timer

Example

```
REM Frame Example
ADDOBJECT "Frame", "Frame",10,10,100,100
Frame.Caption = "My Frame"
ADDOBJECT "OptionButton","Opt",10,20,80,20
Opt.Caption = "My OptionButton"
Opt.parentHWnd = Frame.HWnd
Frame.move 100,100 'move frame and contents
```

Output



Related Items

ADDOBJECT, Events, Methods, Properties

FUNCTION	Statement
----------	-----------

```
FUNCTION procedurename[(arglist)]  
    [statements]  
    [procedurename = expression]  
    [EXIT FUNCTION]  
    [statements]  
    [procedurename = expression]  
END FUNCTION
```

Description

FUNCTION declares a procedure, *procedurename*, that executes *statements*, with *arglist* as parameters, with an optional return value. The required parameter, *procedurename*, is used to call the procedure, and must follow standard variable naming conventions. The optional parameter list, *arglist*, is a comma separated list of variables used to represent variables that are passed to the procedure when it is called. The optional component, *statements*, will be executed as the body of the procedure. Any number of optional EXIT FUNCTION statements can be used to exit the procedure during execution. The value returned by the procedure defaults to EMPTY. To return a value other than the default, assign the value to the function name. The current value of *procedurename* will be returned when the procedure exits by executing all statements or encountering an EXIT FUNCTION statement.

Each member of *arglist* is an argument passed to the procedure, as declared below:

```
[BYVAL | BYREF] varname[()]
```

The optional keyword, BYVAL, specifies that a copy of the variable be passed into the procedure, making its value outside the procedure unmodifiable by the procedure. The optional keyword, BYREF, specifies that the address of the variable be passed into the procedure, making its value outside the procedure modifiable by the procedure. *varname* can be used within the procedure to reference a value passed in, it follows standard variable naming procedure.

Note: If the return value is not stored in a variable or used in an expression, the procedure is called as a SUB procedure, and multiple arguments cannot be enclosed in parenthesis.

Example

```
REM FUNCTION Example  
'FUNCTION: a procedure that returns a value  
DIM Selection, SalePrice  
Selection = Menu("Wednesday")  
PRINT "Wednesday's menu feature:", Selection  
SalePrice = Min(31, 29)
```

```

PRINT "Sale Price:", SalePrice
FUNCTION Menu(day)
  IF day = "Wednesday" THEN
    Menu = "Salisbury Steak"
  END IF
END FUNCTION
FUNCTION Min(x,y)
  IF x > y THEN
    MIN = y
  EXIT FUNCTION
  ELSE
    MIN = x
  END IF
END FUNCTION

```

Output

```

Wednesday's menu feature: Salisbury Steak
Sale Price:      29

```

Related Items

CALL, SUB

GETCOMMANDLINE	Function
-----------------------	-----------------

GETCOMMANDLINE

Description

GETCOMMANDLINE is used to return the command line text that was used to start the program. If the program was started by itself the text returned will be the full path of the program. If the program was started by a document that has been associated with the program, the text returned will be the full path of the program, followed by the full path of the document.

Example

```
REM GETCOMMANDLINE Example
'GETCOMMANDLINE returns launch text
cl = GETCOMMANDLINE
prog = LEFT(cl, INSTR(cl, ".nsb") + 4)
doc = MID(cl, LEN(prog) + 1)
```

Related Items

CURRENTPATH

GETLOCALE **Function**

GETLOCALE
SETLOCALE *localeID*

Description

GETLOCALE returns the current Locale ID. A locale ID determines such things as keyboard layout, alphabetic sort order, date, time number and currency formats.

SETLOCALE sets the locale to *localeID*. *LocaleID* can either be a number or a short string that uniquely identifies a geographical locale. The function returns the old setting of locale ID. Setting *localeID* to 0 will set the locale ID to the current system setting. There is a complete set of Locale IDs here:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/882ca1eb-81b6-4a73-839d-154c6440bf70.asp>

Some common ones are:

1031	de-de	German – Germany
1033	en-us	English – United States
1041	ja	Japanese
2057	en-uk	English - UK

Example

```
PRINT getLocal  
original=SETLOCALE("en-gb") 'Set for England  
PRINT original, getlocale
```

Output

```
1033  
1033 2057
```

Related Items

SETLOCALE, GETLOCALE

GETREF **Function**

GETREF

Description

Returns a reference pointer to a function or subroutine.

Example

```
Function refTest
    MsgBox "RefTest"
End Function

Dim x
Set x=getRef("refTest")
Print TypeName(x)
x
```

Output

Object
RefTest

Related Items

GETRESOURCE **Function**

GETRESOURCE (*Name*[, [*filename*, *register*]])

Description

GetResource allows you to do a number of things to resources contained in your app. The resources must already exist in your app: they must first be added in the IDE as Resources of type 'File' in the Project Explorer of the IDE.

Use this function to:

- Read the contents of a resource into a string in your program.
- Copy the contents of a resource into a file.
- Register and unregister controls.
- Install an ActiveX control and register it.

Name is the name the resource was given in the IDE. It is required to read the contents of a resource or copy into a file.

Filename is the name of the file to copy to or the name of the file to register. If this parameter is not supplied, the content of the resource is returned as a string. If no pathname is supplied, the file is assumed to be in the same directory as your app. If the file already exists, it is not overwritten, and an error message is returned.

Register is True or False. If this parameter is supplied, *Filename* will be registered or unregistered, depending on whether this is true or false.

Example

```
'Extract a string resource
Data = GetResource("MyData")

'Create a data file with data from the
resource
Data = GetResource("MyData", "MyData.txt")

'Create and register an ActiveX control
err = GetResource("Finance", "Finance.dll",
True)

'Unregister an existing control
err = GetResource("", "Finance.dll", False)
```

Output

Related Items

GETSERIALNUMBER	Function
------------------------	-----------------

GETSERIALNUMBER

Description

GETSERIALNUMBER returns the serial number a Pocket PC device, if it has one. They were optional starting with the Pocket PC 2002 and believed to be present in all subsequent devices.

Example

```
REM GETSERIALNUMBER Example
MsgBox GetSerialNumber()
```

Output

(depends on device)

Related Item

GradientButton **Object**

ADDOBJECT "GradientButton", *name*[, *xpos*[, *ypos*
[, *width*[, *height*]]]]

Description

Gradient buttons give apps a modern look and feel. The background color of the button transitions smoothly from one color to another, either vertically or horizontally. It is also useful as a background image. *Name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The optional components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner. Gradient Buttons can only be created using the Device IDE. They cannot be created at runtime.

Properties Supported (see "Properties")

Alignment, BorderStyle, Caption, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, Height, Left, Name, Top, Width

Events Supported (see "Events")

Click

GradientButton Properties

Name	Description
CaptionColor	The color of the text of the button
GradientColor1	The top or left color.
GradientColor2	The bottom or right color
GradientStyle	Top to Bottom or Left to Right

Example

Output



Related Items

COMMANDBUTTON, PICTUREBOX

HEX**Function**

HEX(*number*)

Description

HEX returns a string representation of the hexadecimal (base 16) value of a number. The required parameter, *number*, is any numeric expression. If *number* is not a whole number, it is rounded to the nearest whole number before being converted.

Example

```
REM HEX Example
'HEX returns a number as a hexadecimal string
PRINT "68 in hex:", HEX(68)
PRINT "1 in hex:", HEX(1)
PRINT "2605.45 in hex:", HEX(2605.45)
```

Output

```
68 in hex:      44
1 in hex:       1
2605.45 in hex:      A2D
```

Related Items

OCT

HOURL

Function

HOURL(*time*)

Description

HOURL returns a whole number ranging from 0 to 23 that represents the hour of day of a given time. The required parameter, *time*, can be any numeric or string expression, or any expression that represents a time.

Example

REM HOURL Example

```
'HOURL returns hour of day from a time  
PRINT "The HOURL of " & FormatDateTime(Now,4)  
& _"is " & HOURL(Now)
```

Output

The HOURL of 8/18/1998 10:52:44 PM is 22

(sample date output is system dependant)

Related Items

DAY, MINUTE, MONTH, NOW, SECOND, TIME, YEAR

IF...THEN...ELSE	Statement
-------------------------	------------------

```
IF condition THEN statements [ELSE elstatements]  
IF condition THEN  
    [statements]  
[ELSEIF condition-n THEN  
    [elseifstatements]]...  
[ELSE  
    [elstatements]]  
END IF
```

Description

IF...THEN...ELSE is used to conditionally execute a group of statements. The required component, *condition*, can be any expression that evaluates to TRUE or FALSE. If used inline with no else clause, the *statements* component is required, otherwise, the *statements* component is optional. If *condition* evaluates to TRUE or non-zero, any existing *statements* are executed, if *condition* evaluates to FALSE or zero, execution branches to the next existing ELSEIF clause to evaluate *condition-n*, or to the ELSE clause if it is included.

To execute multiple statements inline, the statements must be separated by a colon (:). If an inline statement consists of a lone procedure call with no arguments, the procedure must be called with empty parenthesis.

Example

```
REM IF...THEN...ELSE Example  
'IF...THEN...ELSE performs conditional  
execution  
DIM Who  
IF TRUE THEN PRINT "TRUE" ELSE PRINT "FALSE"  
IF Who = "Al" THEN  
    PRINT "Big Al"  
ELSEIF Who = "Alien" THEN  
    PRINT "Alien Probe"  
END IF
```

Output

TRUE

Related Items

SELECT CASE

IMP**Function**

result = x IMP y

Description

IMP returns the logical implication of two expressions. Logical implication returns TRUE if x implies y. IMP also does a bitwise comparison of two numeric expressions.

Table 13: Logical Implication

x	y	Implication
TRUE/1	TRUE/1	TRUE/1
TRUE/1	FALSE/0	FALSE/0
FALSE/0	TRUE/1	TRUE/1
FALSE/0	FALSE/0	TRUE/1

Example

```
REM IMP Example
'IMP preforms logical and bitwise implication
DIM Test1, Test2, Test3, x, y
x = 3
y = 5
Test1 = x < 0 IMP y < 10
Test2 = x > 0 IMP y > 10
Test3 = x < 0 IMP y > 10
PRINT "Logical:"
PRINT "  x < 0 IMP y < 10 = " & Test1
PRINT "  x > 0 IMP y > 10 = " & Test2
PRINT "  x < 0 IMP y > 10 = " & Test3
PRINT "Bitwise:"
PRINT "  x IMP y = " & (x IMP y)
```

Output

```
Logical:
  x < 0 IMP y < 10 = True
  x > 0 IMP y > 10 = False
  x < 0 IMP y > 10 = True
Bitwise:
  x IMP y = -3
```

Related Items

AND, EQV, NOT, OR, XOR

INPUTBOX**Function**

INPUTBOX(*prompt*[, *title*[, *default*[, *xpos*, *ypos*]]])

Description

INPUTBOX opens a dialog box to prompt a user to input text or click a button. A string is returned containing the contents of the text field from the dialog box. The required parameter, *prompt*, is a string expression that is displayed in the body of the dialog box. The optional parameter, *title*, is a string expression that is displayed in the title bar. The optional parameter, *default*, is a string expression that is displayed in the text field of the dialog box. The optional parameters, *xpos* and *ypos*, are numeric expressions that specify the horizontal and vertical distance between the upper left corner of the screen and the upper left corner of the dialog box.

If the user taps OK or presses the Enter key, INPUTBOX returns the text in the input field or an empty string ("") if the input field is empty; if the user taps Cancel or presses the escape key (Esc), INPUTBOX returns EMPTY.

InputBox does not work on Pocket PC devices. See the ReadMe file for a workaround.

Example

```
DIM Return
Return = INPUTBOX("Message area", _
    "INPUTBOX Example", "Default text")
PRINT "You entered:", Return
```

Output

You entered: Default Text

Related Items

```
INSTR([start, ]string1, string2[, compare])  
INSTRREV(string1, string2[, start[, compare]])
```

Description

INSTR returns a long value specifying the position of one string within another, measured from the start of the string searched.

INSTRREV returns a long value specifying the position of one string within another, measured from the end of the string searched.

The optional parameter, *start*, is a numeric expression that specifies the starting position of the search within the target string. If *start* is not provided, the search defaults to 1, the first character position for INSTR, or -1, the last character position for INSTRREV. The required parameter, *string1*, is the string being searched. The required parameter, *string2*, is the string that is being searched for. The optional parameter, *compare*, is used to specify the type of search performed. If *string2* is found in *string1*, a positive integer is returned with 0 being the location of the first character, otherwise -1 is returned if *string2* is not found in *string1*.

Example

```
REM INSTR/INSTRREV finds a string in another  
DIM Pos1, Pos2  
Pos1 = INSTR("Cartman", "man")  
Pos2 = INSTRREV("Big Al's Big Boat Ride", _  
    "big", 4, vbTextCompare)  
PRINT "Finding \"man\" from start:", Pos1  
PRINT "Finding \"big\" from end:", Pos2
```

Output

```
Finding "man": 5  
Finding "big": 1
```

Related Items

INT	Function
------------	-----------------

INT(*number*)

Description

INT removes the fractional part of a number, returning the next smallest integer. The required parameter, *number*, is any valid numeric expression.

Example

```
REM INT Example
'INT converts floats to the next smallest int
DIM Pos, Neg
Pos = ATN(1) * 4
Neg = -Pos
PRINT "INT(pi) = " & INT(Pos)
PRINT "INT(-pi) = " & INT(Neg)
```

Output

```
INT(pi) = 3
INT(-pi) = -4
```

Related Items

FIX

result = *object1* IS *object2*

Description

IS returns a boolean specifying if one object reference is identical to another. The required components, *object1* and *object2* are two object references.

Example

```
REM IS Example
DIM Obj2
ADDOBJECT "Finance", "Obj1"
SET Obj2 = Obj1
CompareObjects Obj1, Obj2
SET Obj2 = NOTHING
CompareObjects Obj1, Obj2
SUB CompareObjects(Ob1, Ob2)
    IF Ob1 IS Ob2 THEN
        PRINT "Same"
    ELSE
        PRINT "Different"
    END IF
END SUB
```

Output

Same
Different

Related Items

ADDOBJECT, SET

Is	Function
----	----------

ISARRAY(*expression*)
ISDATE(*expression*)
ISEMPTY(*expression*)
ISNULL(*expression*)
ISNUMERIC(*expression*)
ISOBJECT(*expression*)

Description

The Is functions return TRUE if a variable type corresponds to the function call, FALSE is returned otherwise. The required parameter, *expression*, is the variable whose type status is being determined.

Example

```
REM Is Functions Example
DIM Children(3), Chef, When
TestVariable Children
Chef = 1
TestVariable Chef
When = NOW
TestVariable When
SUB TestVariable(var)
    IF ISARRAY(var) THEN
        PRINT "The variable is an array."
    ELSEIF ISDATE(var) THEN
        PRINT "The variable is a date."
    ELSEIF ISNUMERIC(var) THEN
        PRINT "The variable is a number."
    END IF
END SUB
```

Output

```
The variable is an array.
The variable is a number.
The variable is a date.
```

Related Items

TYPENAME, VARTYPE

JOIN**Function**

JOIN(*stringarray*[, *delimiter*])

Description

JOIN returns a string that is created by concatenating a list of strings with an optional delimit character. The required argument, *stringarray*, is a one-dimensional array of strings. The optional argument, *delimiter*, is a string expression whose first character is used to separate the strings joined from *stringarray*; if no string is provided a space character (" ") is used by default. To concatenate with no delimiters, use an empty space ("") as the delimiter.

Example

```
REM JOIN Example
'JOIN concatenates strings
DIM Words, Letters
Words = ARRAY("Hello", "World")
Letters = ARRAY("D", "a", "l", "l", "a", "s")
PRINT JOIN(Words) & "!"
PRINT JOIN(Letters, "")
```

Output

```
Hello World!
Dallas
```

Related Items

SPLIT

KeyboardStatus**Global**

KeyboardStatus=False|True
KeyboardStatusChanged

Description

KeyboardStatus is a global variable that sets or gets whether the on screen keyboard appears on Windows Mobile devices. If it is False, then no keyboard is shown. A value of True means the on screen keyboard is present.

If the state of the on screen keyboard is changed, the KeyboardStatusChanged event is sent to the program

Example

```
REM KeyboardStatus Example
PRINT KeyboardStatus
IF KeyBoardStatus=False THEN
KeyBoardStatus=True
ELSE KeyBoardStatus=True
SUB keyboardStatusChanged
    MSGBOX "Keyboard Status Changed"
END SUB
```

Output

(keyboard appears or disappears from view)

Related Items

KeyPreview = TRUE | **FALSE**

Description

KeyPreview is a global property that enables the Output object to receive all keystroke events. KeyPreview is FALSE by default, which prevents the Output object from receiving any keystroke events. Setting KeyPreview to TRUE allows the Output to receive all keystroke events, regardless of which object is activated to receive keyboard input.

Example

```
REM KeyPreview Example
'KeyPreview enables keystroke events in Output
KeyPreview = TRUE
AddObject "TextBox", "Text", 50, 50, 90, 90
SUB Output_KeyPress(key)
  PRINT "Key Pressed: ", key
END SUB
```

Output**Related Items**

KeyDown, KeyPress, KeyUp, PictureBox, SetFocus, Output

KILLFOCUS	Statement
------------------	------------------

KillFocus

Description

KillFocus removes the focus from the current object.

Example

```
REM KillFocus Example
AddObject "TextBox","TB",10,10,20,20
' set focus to TextBox
tb.SetFocus
' remove focus from Textbox
KillFocus
```

Output

(cursor is placed in TextBox, then disappears)

Related Items

<u>Label</u>	<u>Object</u>
---------------------	----------------------

ADDOBJECT "Label", *name*, *xpos*, *ypos*, *width*, *height*

Description

Label is used to display read-only text in an object that floats over the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object. The optional components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner.

Note: By default, the TabStop property is FALSE preventing activation for keyboard input from the user interface.

Properties Supported (see "Properties")

Alignment, BackColor, BorderStyle, Caption, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Hwnd, Left, Name, ParentHwnd, TabStop, Tag, Text, Timer, Top, UseMnemonic, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Click, Timer

Example

```
REM Label Example
ADDOBJECT "Label", "Label", 10, 10, 40, 20
Label.BackColor = Output.BackColor
ADDOBJECT "TextBox", "Input", 55, 10, 160, 20
Input.text = ""
Input.SetFocus
```

Output

Related Items

ADDOBJECT, Events, Methods, Properties, TextBox

LBOUND	Function
---------------	-----------------

LBOUND(*array*[, *dimension*])

Description

LBOUND returns a long value specifying the smallest available subscript in *dimension* of an array, *array*. The required parameter, *array*, is any array variable. The optional parameter, *dimension*, specifies which dimension's lower bound is returned, beginning with the default, 1. The lower bound of any dimension of an array is always 0.

Example

```
REM LBOUND Example
'LBOUND returns lower bound of array dimension
DIM Other, Children(3), Parents(3, 1)
Other = ARRAY("Damien", "Pip", "Wendy")
PRINT "'Other' Lower Bound:", LBOUND(Other)
PRINT "'Children' Lower Bound:", _
    LBOUND(CHILDREN, 1)
PRINT "'Parents' Lower Bounds:", _
    LBOUND(Parents), LBOUND(Parents, 2)
```

Output

```
'Other' Lower Bound:      0
'Children' Lower Bound:   0
'Parents' Lower Bounds:   0      0
```

Related Items

ARRAY, DIM, REDIM, UBOUND

LCASE	Function
--------------	-----------------

LCASE(*string*)

Description

LCASE returns string with all of its uppercase characters converted to lowercase. The required parameter, *string*, is any valid string expression.

Example

```
REM LCASE Example
'LCASE returns string with all lowercase chars
DIM Uncle
Uncle = "JIM"
PRINT Uncle & " lowercase is " & LCASE(Uncle)
```

Output

JIM lowercase is jim

Related Items

UCASE

LEFT**Function**

LEFT(*string*, *length*)

LEFTB(*string*, *length*)

Description

LEFT returns a string containing a specified number of characters from the left end of a string. The required parameter, *string*, is any valid string expression. The required parameter, *length*, is any valid numeric expression, if *length* is 0, an empty string ("") is returned, if *length* is greater than the size of string, the entire string is returned.

LEFTB is a strict bitwise version of LEFT. When used on the Unicode strings of Windows CE devices, it will return any partial character as a normal character.

Example

```
REM LEFT Example
'LEFT returns substring from string left end
DIM Wendy, Eric
Wendy = "Testaburger"
Eric = "Cartman"
PRINT "The LEFT 4 of " & Wendy & ": " _
      & LEFT(Wendy, 4)
PRINT "The LEFT 4 of " & Eric & ": " _
      & LEFT(Eric, 4)
```

Output

```
The LEFT 4 of Testaburger: Test
The LEFT 4 of Cartman: Cart
```

Related Items

LEN, MID, RIGHT

LEN**Function**

LEN(*string* | *variable*)
LENB(*string* | *variable*)

Description

LEN returns a long value specifying the number of characters in a string, or the number of bytes required to output a variable. The optional parameter, *string*, is any valid string expression. The optional parameter, *variable*, is any variable, if *variable* contains a string the length of the string is returned. One (and only one) of the optional parameters, *string* and *variable*, must be supplied.

LENB is a strict bitwise version of LEN. LENB always returns a measurement in bytes. When used on the Unicode strings of Windows CE devices, it will return a number twice the value returned by LEN.

Example

```
REM LEN Example
'LEN returns string length or variable size
DIM Frog, Survived
Frog = "Staring"
Survived = 2
PRINT "LEN of Frog:", LEN(Frog)
PRINT "LENB of Frog:", LENB(Frog)
PRINT "LEN of Survived:", LEN(Survived)
```

Output

```
LEN of Frog:      7
LENB of Frog:    14
LEN of Survived:      1
```

Related Items

ADDOBJECT "ListBox", *name*, *xpos*, *ypos*, *width*, *height*

Description

ListBox is used to display a text list object that floats over the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object. It must follow standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner. The ListIndex property is the index of the selected item in the list, the index of the first item is 0; if no item is selected, the ListIndex property is -1. MultiSelect must be set to True before any items are added to the listbox.

Properties Supported (see "Properties")

BackColor, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Hwnd, IntegralHeight, Left, List (read only), ListCount (read-only), ListIndex, MultiSelect, Name, NewIndex, ParentHwnd, Redraw, ScrollBars, SelCount, Selected, SpinBox, Sorted, Tag, TabStop, Text, Timer, Top, TopIndex, Visible, Width, WindowLong

Methods Supported (see "Methods")

AddItem, Clear, Hide, Move, RemoveItem, SetFocus, Show

Events Supported (see "Events")

Change, Click, DblClick, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timer

Example

```
REM ListBox Example
'ListBox is a text list object
DIM i
ADDOBJECT "ListBox", "List", 10, 10, 100, 100
List.ScrollBars = 2
FOR i = vbSUNDAY TO vbSATURDAY
    List.AddItem WEEKDAYNAME(i)
NEXT
List.ListIndex = WEEKDAY(NOW-1)
List.SetFocus
```

Output**Related Items**

ADDOBJECT, ComboBox, Events, Methods, Properties

LOG**Function**

LOG(*number*)

Description

LOG returns a double-precision value equal to the natural logarithm of a number. The required parameter, *number*, is any numeric expression. The natural logarithm is the base e logarithm; e is approximately equal to 2.718282.

Calculating base-n logarithm of x is achieved by dividing the natural logarithm of x by the natural logarithm of n.

Example

```
REM LOG Example
'LOG calculates natural logarithms
DIM e
e = 2.718282
PRINT "LOG(1) = " & LOG(1)
PRINT "LOG(e) = " & LOG(e)
PRINT "LOG10(2) = " & LogN(10, 2)
FUNCTION LogN(Base, Number)
    LogN = LOG(Number) / LOG(Base)
END FUNCTION
```

Output

```
LOG(1) = 0
LOG(e) = 1
LOG10(2) = 0.30103
```

Related Items

EXP

LTRIM	Function
--------------	-----------------

LTRIM(*string*)

Description

LTRIM returns a string with all leading spaces removed. The required parameter, *string*, is any valid string expression.

Example

```
REM LTRIM Example
'LTRIM trims all leading spaces
DIM Spacey
Spacey = "-K"
PRINT "(" & Spacey & ")"
PRINT "(" & LTRIM(Spacey) & ")"
```

Output

```
(-K)
(K)
```

Related Items

RTRIM, TRIM

Methods**Object**

ObjectName.Method [arglist]

Description

Object methods are procedures in objects. The required component, *ObjectName*, is a valid object expression that refers to an object added to a program with an ADDOBJECT statement. The required component, *Method*, is the name of a FUNCTION or SUB procedure in an object, see below for a list of common object methods. The optional component, *arglist*, is a comma-separated list of values that are passed to the procedure as arguments. Calling a method in an object can change how an object is displayed, how it interacts, and the values of its properties.

Table 14: Object Methods

Method	Arguments	Comments
AddItem	text[, index]	Adds <i>text</i> to the list of items, with an optional <i>index</i> . If no <i>index</i> is supplied, <i>text</i> is put at the end of unsorted lists, or <i>text</i> is inserted to proper place in sorted lists. ComboBox, ListBox
Clear		Clears all items from list. ComboBox, ListBox
Hide		Hide object and set Visible property to FALSE.
Move	x, y, w, h	Moves object to new x, y location, with optional resizing if w and h are provided.
RemoveItem	index	Remove an item from list by <i>index</i> (0 = first item). ComboBox, ListBox
SetFocus		Activate object to receive keyboard input.
Show		Show object and set Visible property to TRUE.

Example

```
REM Object Methods Example
'Methods are procedures in objects
ADDOBJECT "ComboBox", "Combo", 5, 5, 150, 110
Combo.AddItem("Eric")
```

```
Combo.AddItem("Kenny")  
Combo.AddItem("Kyle")  
Combo.AddItem("Stan")  
Combo.ListIndex = 0  
Combo.SetFocus
```

Output**Related Items**

Events, Properties

MID	Function
-----	----------

MID(*string*, *start*[, *length*])
MIDB(*string*, *start*[, *length*])

Description

MID returns a string containing characters from the middle of a string. The required parameter, *string*, is any valid string expression. The required parameter, *start*, is any valid numeric expression, if *start* is greater than the length of string, a zero-length string is returned (""). The optional parameter, *length*, is any valid numeric expression and it specifies the number of characters to return. If *length* is not used, or exceeds the number of remaining characters in string, all characters from start to the end of string are returned.

MIDB is a strict bitwise version of MID. When used on the Unicode strings of Windows CE devices, it will return any partial character as a normal character.

Example

```
REM MID Example
'MID returns substring from string middle
DIM Eric, Mister
Eric = "Cartman"
PRINT "From Cartman:", MID(Eric, 2, 3)
Mister = "Hankey"
PRINT "From Hankey:", MID(Mister, 4)
```

Output

```
From Cartman: art
From Hankey:   key
```

Related Items

LEFT, RIGHT

MINUTE**Function**

MINUTE(*time*)

Description

MINUTE returns a whole number ranging from 0 to 59 that represents the minute of the hour, of a given time. The required parameter, *time*, can be any numeric or string expression, or any expression that represents a time.

Example

```
REM MINUTE Example
'MINUTE returns minute of hour from a time
DIM When
When = NOW
PRINT "The MINUTE of " & When & " is " _
      & MINUTE(When)
```

Output

The MINUTE of 8/18/1998 10:52:44 PM is 52

(sample date output is system dependant)

Related Items

DAY, HOUR, MONTH, NOW, SECOND, TIME, YEAR

MOD**Operator**

result = *x* MOD *y*

Description

MOD divides *x* by *y* and returns the integer part of the remainder. The required parameters, *x* and *y*, are any valid numeric expressions. The value of *result* is a whole number with a magnitude less than the magnitude of *y*.

Example

```
REM MOD Example
'MOD returns quotient remainder as integer
DIM Answer
Answer = 15 MOD 2
PRINT "15 MOD 2 = " & Answer
Answer = 21 MOD 3.7
PRINT "21 MOD 3.7 = " & Answer
```

Output

```
15 MOD 2 = 1
21 MOD 3.7 = 1
```

Related Items

MONTH**Function**

MONTH(*date*)

Description

MONTH returns a whole number ranging from 1 to 12 that represents the month of the year, of a given date. The required parameter, *date*, can be any numeric or string expression, or any expression that represents a date.

Example

REM MONTH Example

'MONTH returns month of year of a date

DIM When

When = NOW

PRINT "The MONTH of " & When & " is " _
 & MONTH(When)

Output

The MONTH of 8/18/1998 10:52:44 PM is 8

(sample date output is system dependant)

Related Items

DAY, HOUR, MINUTE, NOW, SECOND, TIME, YEAR

MONTHNAME	Function
------------------	-----------------

MONTHNAME(*month* [, *abbreviate*])

Description

MONTHNAME returns the string name of the given month. The required parameter, *month*, is a numeric expression ranging from 1 to 12. The optional parameter, *abbreviate*, is a boolean variable that specifies whether MONTHNAME returns the complete month name or its three-letter abbreviation.

Example

```
REM MONTHNAME Example
PRINT MONTHNAME (MONTH (NOW) )
PRINT MONTHNAME (12, TRUE)
```

Output

```
August
Dec
```

Related Items

WEEKDAYNAME

MSGBOX**Function**

MSGBOX(*prompt*[, *buttons*[, *title*]])

Description

MSGBOX opens a dialog box, and waits for the user to tap on a button. The return value is an integer that indicates which button was tapped. The required parameter, *prompt*, is a string expression that is displayed in the body of the dialog box. The optional parameter, *buttons*, is a numeric or constant expression that specifies which buttons to draw, which icon to use, the identity of the default button, and the modality of the dialog box. The default value for buttons is 0, other values are obtained by adding the desired constants from table below. The optional parameter, *title*, is a string expression that is displayed in the title bar of the dialog box.

Some Windows CE devices do not have this function.

Table 15: Button constants

Constant	Value	Description
vbOKOnly	0	OK Button only
vbOKCancel	1	OK and Cancel buttons
vbAbortRetryIgnore	2	Abort, Retry, and Ignore buttons
vbYesNoCancel	3	Yes, No, and Cancel buttons
vbYesNo	4	Yes and No buttons
vbRetrvCancel	5	Retrv and Cancel buttons
vbCritical	16	Critical Message icon
vbQuestion	32	Warning Querv icon
vbExclamation	48	Warning Message icon
vbInformation	64	Information Message icon
vbDefaultButton1	0	First button is default
vbDefaultButton2	256	Second button is default
vbDefaultButton3	512	Third button is default
vbDefaultButton4	768	Fourth button is default
vbApplicationModal	0	Application interface is disabled until user responds to dialog box
vbSystemModal	4096	Application and system interfaces are disabled until user responds to dialog box.

Table 16: MSGBOX return values

Constant	Value	Description
vbOK	1	OK
vbCancel	2	Cancel
vbAbort	3	Abort
vbRetrv	4	Retrv
vbIgnore	5	Ignore
vbYes	6	Yes
vbNo	7	No

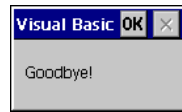
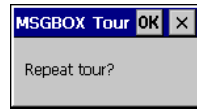
Example

```

REM MSGBOX Example
'MSGBOX displays a modal dialog box
CONST TITLE = "MSGBOX Tour"
DIM Continue
MSGBOX("Hello World!")
MSGBOX "Brief tour of MSGBOX", 0, TITLE
Continue = MSGBOX("Continue tour?", _
    vbYesNo + vbQuestion, TITLE)
IF Continue = vbYes THEN
    Continue=MSGBOX("Short tour, huh?", _
        vbInformation + vbYesNo, TITLE)
END IF
Continue = MSGBOX("Repeat tour?", 257, TITLE)
IF Continue = vbCancel THEN
    MSGBOX("Goodbye!")
END IF

```

Output



Related Items
INPUTBOX

NOT**Operator**

result = NOT *expression*

Description

NOT returns the logical negation of an expression. The required argument, *expression*, is any valid expression. *result* is TRUE if *expression* evaluates to FALSE, FALSE if *expression* evaluates to TRUE, *result* is NULL otherwise.

NOT also does a bitwise inversion of an expression. Each bit in *result* is set to 1 if the corresponding bit in *expression* is 0, otherwise it is set to 0.

Example

```
REM NOT Example
'NOT does logical negation & bitwise inversion
DIM Test1, Test2, Test3, x, y
x = 3
y = 8
Test1 = NOT(x > 0)
Test2 = NOT(y > 10)
Test3 = NOT x
PRINT "Logical:"
PRINT "  NOT(x > 0) = " & Test1
PRINT "  NOT(y > 10) = " & Test2
PRINT "Bitwise:"
PRINT "  NOT x = " & Test3
```

Output

```
Logical:
  NOT(x > 0) = False
  NOT(y > 10) = True
Bitwise:
  NOT x = -4
```

Related Items

AND, EQV, IMP, OR, XOR

NOW	Function
------------	-----------------

NOW

Description

NOW returns the current date and time according to the date and time setting of your computer.

Example

```
REM NOW example
PRINT FormatDateTime (Now)
```

Output

```
11/11/2004 9:52:27 AM
```

Related Items

Date, Day, Hour, Minute, Month, Second, Time, Weekday, Year

NSBVERSION**Global**

NSBVERSION

Description

NSBVersion returns a string with the current version of NS Basic/CE that is running. This global property can be read, but not set.

Example

```
REM NSBVersion Example
PRINT "The current version of NS Basic is "
& NSBVersion
```

Output

The current version of NS Basic is v. 4.0.1

Related Items

OCT**Function**

OCT(*number*)

Description

OCT returns a string representation of the octal (base 8) value of a number. The required parameter, *number*, is any valid numeric expression. If *number* is not a whole number, it is rounded to the nearest whole number before being converted.

Example

```
REM OCT Example
'OCT returns a number as an octal string
PRINT "68 in octal:", OCT(68)
PRINT "1 in octal:", OCT(1)
PRINT "2605.45 in octal:", OCT(2605.45)
```

Output

```
68 in octal:   104
1 in octal:    1
2605.45 in octal: 5055
```

Related Items

HEX

ON ERROR	Statement
ON ERROR RESUME NEXT	
ON ERROR GOTO 0	

Description

ON ERROR is used to allow error-handling in procedures, by catching fatal run-time errors and enabling continued execution. In the first form, RESUME NEXT, execution continues with the statement immediately following the statement where the error occurred. The second form, GOTO 0, is used to disable error-handling in the current procedure. If ON ERROR statements aren't used, any run-time error is fatal, displays an error message, and terminates execution.

Example

```
REM ON ERROR Example
'ON ERROR does error-handling in procedures
ADDOBJECT "File"
GetToDoList
SUB GetToDoList
    ON ERROR RESUME NEXT
    File.OPEN "ToDo", 1
    IF ERR.NUMBER THEN
        PRINT "Delayed ERROR handling"
        PRINT "ERROR Source: " & ERR.Source
    END IF
    ON ERROR GOTO 0
END SUB
```

Output

```
Delayed ERROR Handling
ERROR Source: File
```

Related Items

Err Object

OPTION EXPLICIT	Statement
------------------------	------------------

OPTION EXPLICIT

Description

OPTION EXPLICIT is used at script level, to force explicit declaration of all variables. When used, OPTION EXPLICIT must appear before any FUNCTION or SUB procedures, and any undeclared variables will cause an error to occur. Use DIM or REDIM to declare variables.

OPTION EXPLICIT can improve program performance significantly, and enforces good programming practices. It should be used whenever possible.

If you want to use STEP or TRACE, OPTION EXPLICIT must be the very first line of your program.

Example

```
OPTION EXPLICIT
DIM Teacher
Teacher = "Mr. Garrison"
```

Related Items

DIM, REDIM

OptionButton	Object
---------------------	---------------

ADDOBJECT "OptionButton", *name*, *xpos*, *ypos*,
width, *height*

Description

OptionButton is used to display a button object with a circular toggle that floats over the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object following standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner. The Value property is FALSE, when the circle is empty; the Value property is TRUE, when the circle is full.

To group several OptionButtons into an exclusive group, often called RadioButtons, use the Group property. Setting Group to True starts a new group. Subsequent OptionButtons with Group set to false will become part of that same group, in order of creation.

Properties Supported (see "Properties")

Alignment, BackColor, Caption, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Group, Height, Hwnd, Left, Name, ParentHwnd, TabStop, Tag, Text, Timer, Top, Value, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Click, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timerf

Example

```
REM OptionButton Example
'OptionButton is a toggle button with a circle
'often referred to as a Radio Button
ADDOBJECT "OPTIONBUTTON", "O1", 10, 10, 100, 20
O1.CAPTION="FEMALE"
O1.VALUE=TRUE
ADDOBJECT "OPTIONBUTTON", "O2", 10, 30, 100, 20
O2.CAPTION="MALE"
O2.GROUP=FALSE
ADDOBJECT "OPTIONBUTTON", "O3", 10, 50, 100, 20
O3.CAPTION="UNDISCLOSED"
O3.GROUP=FALSE
```

OUTPUT

- ☒ Female
- ☐ Male
- ☐ Undisclosed

Related Items

ADDOBJECT, CheckBox, Events, Methods, Properties

OR**Operator**

result = x OR y

Description

OR returns the logical disjunction of two expressions. *result* is TRUE, if one or both expressions x and y evaluate to TRUE, otherwise, *result* is FALSE.

OR also does a bitwise comparison of two numeric expressions. Each bit in *result* is set to 1 if either corresponding bit in x or y is 1, otherwise it is set to 0.

Example

```
REM OR Example
'OR performs logical and bitwise disjunction
DIM Test1, Test2, Test3, x, y
x = 1
y = 5
Test1 = x > 0 OR y < 10
Test2 = x > 0 OR y > 10
Test3 = x OR y
PRINT "Logical:"
PRINT "  x > 0 OR y < 10 = " & Test1
PRINT "  x > 0 OR y > 10 = " & Test2
PRINT "Bitwise:"
PRINT "  x OR y = " & Test3
```

Output

```
Logical:
  x > 0 OR y < 10 = True
  x > 0 OR y > 10 = True
Bitwise:
  x OR y = 5
```

Related Items

AND, EQV, IMP, NOT, XOR

`Output.property=value`

Description

The Output object is created automatically when you run an NS Basic/CE program and is the bottom most object. It is a PictureBox object, so you can use all the same properties, methods and events. In addition, output from the PRINT statement goes to Output.

When the Output object is closed using the Close box in the top right corner, a `Output_close` event is sent to your program. You may use this to do limited cleanup work.

Properties Supported (see "Properties")

Same as PictureBox.

Methods Supported (see "Methods")

Same as PictureBox.

Events Supported (see "Events")

Same as PictureBox, plus `Output_Close`, `Output_Size`

Example

```
REM Show close box action
SUB form_close
    MSGBOX "The Output box is closing"
END SUB
```

Output**Related Items**

ADDOBJECT, Events, Methods, Properties, PictureBox, KeyPreview

PictureBox**Object**

```
ADDOBJECT "PictureBox"[, name[, xpos[, ypos  
[, width[, height]]]]]
```

Description

PictureBox is used to display text and graphics in objects that float over the output window. The optional component, *name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The optional components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner. The output window itself is a PictureBox, the program variable Output can be used with PictureBox properties, methods, and events.

Properties Supported (see "Properties")

BackColor, BorderStyle, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, Left, Name, Tag, Text, Top, Width

Table 17: PictureBox Properties

Name	Description
DrawWidth	Line width for draw methods
FillColor	Color, used to fill shapes, circles or boxes.
FillStyle	0 solid, 1 transparent
FontTransparent	True/False
Picture	String with name of an image file.
ScaleHeight	Units for height using custom scale
ScaleLeft	Left edge of object, to scale
ScaleMode	0 user defined 1 Twips (1440 dpi) 2 Points (72 dpi) 3 Pixels 4 Characters (horizontal = 120 twips, vertical = 240 twips) 5 inches 6 millimeters 7 centimeters
ScaleTop	Top of current object
ScaleWidth	Units for width using custom scale
Tag	Use for anything

Methods Supported (see "Methods")

Move

Table 18: PictureBox Methods

Name	Arguments	Description
Cls		Clears text and graphics, but not print.
DrawCircle	x, y, radius[, color [, aspectRatio]]	Draw a circle or ellipse.
DrawLine	x1, y1, x2, y2[, color [, box [, fill]]]	box is TRUE/FALSE
DrawPicture	filename, x1, y1[, width, height, x2, y2, width2, height2], rasterop]	Draw part or all of an image file on the object (can also rescale).
DrawPoint	x, y[, color]	Draw a single point.
DrawText	Text[.x,y]	Draws on next line on object., or at x,y if specified.
Refresh		Redraw the object.
ScaleX	width, from, to	Converts scale of width.
ScaleY	height, from, to	Converts scale of height.
SetScale	x1, y1, x2, y2	Use to change scale.
TextHeight	text	Returns height of text., embedded CR's OK.
TextWidth	text	Returns width of widest line.

Events Supported (see "Events")

Click

Table 19: PictureBox Events

Name	Arguments	Description
KeyDown†	keyCode+, shift*	Key is pressed.
KeyPress†	keyCode+	Combination of KeyDown and KeyUp.
KeyUp†	keyCode+, shift*	Key is released.
MouseDown#	button, shift*, x, y	Object is contacted with stylus or input device.

MouseMove#	button, shift*, x, y	Stylus or input device is moved while in contact with object.
MouseUp#	button, shift*, x, y	Stylus is lifted from object, or input device breaks contact with object.

+ Keyboard code of key that generated event.
 * 1 = Shift key, 2 = CTRL key, 4 = ALT key
 † Only in Output PictureBox object. Global property KeyPreview must be TRUE to receive this event.
 # button is input device button used to contact object (0 = Stylus). x, y, show location of contact relative to object

Example

```

REM PictureBox Example
ADDOBJECT "PictureBox", "PBox", 65, 0, 55, 20
PBox.BorderStyle = 1
PBox.BackColor = vbWHITE
PBox.DrawText " Button"
SUB PBox_Click
  PRINT Output.Width & " X " & Output.Height
END SUB
  
```

Output

(screen size is machine dependant)

Related Items

ADDOBJECT

PLAYSOUND Function

PLAYSOUND(*file*, *hmod*, *flags*)

Description

The PlaySound function plays a sound specified by the specified file, resource, or system event. *File* is the name of the file to play: using 0 in this argument will stop the current sound. *Hmod* is usually 0 - if you are using a resource file, it is the handle to the resource. *Flags* is formed by adding the following flags:

Table 20

Flag	Value	Description
APP	?	Play using specific App
ALIAS	&h00010000	<i>File</i> is a system-event alias in the registry or the WIN.INI file. Not for FILENAME or RESOURCE.
ASYNCH	&h00000001	Return after sound begins.
FILENAME	&h00020000	<i>File</i> is a filename.
LOOP	&h00000008	Play until PlaySound is called with <i>file</i> set to 0.
MEMORY	&h00000004	<i>File</i> points to an image of a sound in memory.
NODEFAULT	&h00000002	If sound cannot be found, return silently.
NOSTOP	&h00000010	Yield to another sound.
NOWAIT	&h00002000	Return if driver busy.
RESOURCE	&h00040004	<i>File</i> is a resource ID; <i>Hmod</i> is the instance.
SYNC	&h00000000	PlaySound returns after the sound completes

Example

```
PlaySound "SystemStart",0,&h10000
Sleep 500
PlaySound 0,0,0 'stop it after .5 secs
```

Output

(sound plays for half a second)

Related Items

WAVEVOLUME

PRINT	Statement
--------------	------------------

PRINT [*expressionA*[, *expressionB*[, *expressionC*[, ...]]]]

Description

PRINT writes text to the Output window. PRINT writes up to 20 comma-delimited expressions, separating each with a tab and appending a carriage return. When used without expressions, PRINT, writes a blank line to the Output window.

You will not see the output from this statement if you are using forms, since the forms cover the Output window completely.

Example

```
REM PRINT Example
'PRINT writes text to the output window
PRINT "Hello World!"
PRINT
PRINT "The time is", NOW
```

Output

Hello World!

The time is 8/18/1998 10:52:44 PM

Related Items

Properties**Object**

ObjectName.Property[= *value*]

Description

Object properties are variables in objects. The required component, *ObjectName*, is a valid object expression that refers to an object added to a program with an ADDOBJECT statement. The required component, *Property*, is the name of a variable in an object. See below for a list of common object properties. The optional component, *value*, is assigned to the variable in the object. Changing a property of an object can change how an object is displayed and how it interacts.

Some properties affect the fundamental way an object appears, and should be set directly after the object is created.

Some properties are read-only and can only be set at design time. Not all properties are available at runtime to be read.

Table 21: Properties

Property Name	Legal Values	Comments
Alignment+	Integer 0 = Left Justify 1 = Right Justify 2 = Center Justify	Text/Caption text alignment. Label, OptionButton
BackColor	Color (Integer)	Background color of object. Colors may be RGB values or the index of a color in the system palette.
BorderStyle	0 = none 1 = line	Date, Label, TextBox, Time
Bottom	Integer	Obsolete – use Height
Caption	String	Text displayed in object.
Date	Date type	Set the date, i.e. Date.date=cdate("8/5/02")

Property Name	Legal Values	Comments
Default	true/FALSE	Sets a button as the default option chosen when the enter key is used.
Dir To Include	Pathname	A comma separated list of directories of files to be installed with your app.
Enabled	True	Does object respond to user events?
Encrypted	TRUE/FALSE	Project Property only. If set, saved file will be encrypted. Desktop IDE only.
ExpandedHeight	Integer	Size of combobox when expanded.
FontBold	TRUE/FALSE	
FontItalic	TRUE/FALSE	
FontName	String	
FontSize	Integer	
FontStrikethru	true/FALSE	
FontUnderline	true/FALSE	
FontWeight	integer, 400 = normal 700 = bold	values greater than 550 are converted to 700 all others are converted to 400
ForeColor	Color (Integer)	Foreground color of object for text and drawing. Colors may be RGB values or the index of a color in the system palette.
Group	TRUE/false	Start a new option group. OptionButton.
Height	Integer	Pixels from top edge to bottom edge of object

Property Name	Legal Values	Comments
Hi Res Aware	True/ False	If True, will take advantage of Hi Res devices.
HideSelection	TRUE/false	Causes the selected text to hide or show when the control has lost focus. TextBox
Hwnd†	Long	Internal reference to object
Icon†	Path	Icon for the app, in .ico format. Must contain 32x32 and 16x16 images.
IntegralHeight+	TRUE/FALSE	Constrain object to a height that is an integral multiple of individual line height. ComboBox, ListBox
Left	Integer	Pixels from left edge of output window to left edge of object
List(index†)	string	Array of values. Index argument required†
ListCount†	Integer	Number of items in list. ComboBox, ListBox
ListIndex	Integer	Currently selected item. ComboBox, ListBox
Locked	true/FALSE	Enables or disables changes to text, while not affecting the ability to select text. TextBox
LongFormat	true/FALSE	format to display date. Date
LowercaseOnly	true/FALSE	Constrains text entered to only lowercase characters. ComboBox, TextBox

Property Name	Legal Values	Comments
MaxLength	Integer (0-30,000)	Number of characters allowed. TextBox (multiline)
MultiLine+	TRUE/FALSE	Enable embedded carriage returns and text wrapping. TextBox
MultiSelect	true/FALSE	Allows multiple items to be selected. ListBox
Name†		The name of the object
NewIndex	Integer	Insertion index for for AddItem in unsorted list objects. ComboBox, ListBox
NumbersOnly	true/FALSE	Constrains text entered to only numeric characters. TextBox
ParentHWnd	Window Handle	The parent of an object
Password	true/FALSE	Hides text input, visually replacing all characters with an asterisk (*). TextBox
Redraw	TRUE/false	Enable or disable screen updates while adding multiple items to list. ComboBox, ListBox
Right	Integer	Obsolete – use Width
Scrollbars+	Integer 0 = No scroll bar 1 = Horizontal 2 = Vertical 3 = Both	ListBox (vertical only), TextBox (multiline)
SelCount	Integer	Read only. Returns the number of items selected. ListBox
Selected(index)	TRUE/FALSE	Runtime only. Set or check selection

Property Name	Legal Values	Comments
		state of item <i>index</i> in the list. ListBox
SelLength	Integer	The number of characters selected. Textbox
SelStart	Integer	The first character selected. Textbox
SelText	String	The selected string. TextBox
Sorted+	TRUE/FALSE	Items added with AddItem are inserted alphabetically. ListBox, ComboBox
SpinBox+	TRUE/FALSE	If TRUE, will rotate values on one line. Listbox only.
Style+	Integer 0 = editable list 2 = read-only list	Input line is editable/read-only. ComboBox
Tabstop	TRUE/FALSE*	Allows object to activate to receive keyboard input by tabbing.
Tag	String	Available for your use
Text	String	Text displayed in object.
Timer	Long	Event <i>object_timer</i> fires after this number of milliseconds. 0 to disable.
Top	Integer	Pixels from top edge of output window to top edge of object
TopIndex	Integer	Get or set which item is displayed in the topmost position. ComboBox, ListBox
UppercaseOnly	true/FALSE	Constrains text entered to only uppercase

Property Name	Legal Values	Comments
		characters. ComboBox, TextBox
UseMnemonic	TRUE/false	The underscore character can be added to window caption with an ampersand (&). Label
Value	TRUE/FALSE or 0/1	Object is checked or selected. CheckBox, OptionButton
Visible†	TRUE/FALSE	Set with Hide/Show methods.
Width	Integer	Pixels from left edge to right edge of object
WindowLong(index)	Index: 0=ExWindowStyle 1=WindowStyle	See Microsoft Windows docs for SetWindowLong

† read-only property. Set at design time only.

+ affects fundamental object display

* default is FALSE for Label object

Example

```
REM Object Property Example
'A property is a variable in an object
ADDOBJECT "TextBox", "Input" 50, 50, 100, 100
ADDOBJECT "CommandButton", "Button", 150,
170, 20, 80
PRINT "Text: " & Button.Text
Button.Text = "Push Me!"
PRINT "Visible: " & CBOOL(Input.Visible)
Input.Hide
PRINT "Visible: " & CBOOL(Input.Visible)
```

Output

```
Text:
Visible: True
Visible: False
```

Related Items

Events, Methods

RANDOMIZE**Statement**

RANDOMIZE [*number*]

Description

RANDOMIZE initializes the random-number generator. The optional parameter, *number*, is any valid numeric expression that is used as a seed for the RND function. If *number* is omitted, the value of the system timer is used as the seed value. To repeat a random-number sequence, call RND with a negative argument, then call RANDOMIZE with a numeric argument; calling RANDOMIZE alone with the same value for *number* will not repeat a sequence.

Example

```
REM RANDOMIZE Example
'RANDOMIZE initialize random-number generator
RANDOMIZE
Random
RANDOMIZE 44
Random
RND -1
RANDOMIZE 169
Random

SUB Random
  DIM Ret
  RET = ""
  FOR i = 1 to 4
    RET = RET & INT((RND * 1000) + 1) & " "
  NEXT
  PRINT "Four random numbers:", RET
END SUB
```

Output

```
Four random numbers:  8 912 43 537
Four random numbers: 33 6 430 51
Four random numbers: 36 6 192 54
```

Related Items

RND

REDIM	Statement
--------------	------------------

REDIM [PRESERVE] *nameA*[(*subscriptA*[, *subscriptB*[,
 subscriptC...]])][, *nameB*...[, *nameC*...[...]]]

Description

REDIM is used to reallocate storage space for fixed arrays. The required component, *name*, is the name of the variable, and it must follow standard variable naming conventions. The optional list of *subscript* arguments are the dimensions of an array variable, up to 60 comma separated, numeric expressions can be used. The optional keyword, PRESERVE will preserve data in an existing array, when only the size of the last dimension is changed. The lower bound of an array is always 0.

Example

```
REM REDIM Example
'REDIM reallocates array storage
'An empty variable named "Foo"
DIM Foo
'A one-dimensional array with 10 elements
REDIM Foo(9)
'A two-dimensional array with 600 elements
'20 x 30
REDIM Foo(19, 29)
```

Related Items

ARRAY, DIM

REM	Statement
------------	------------------

REM *remarks*

'remarks

Description

REM is used to include remarks, or comments, as text which is never executed, in a program. The optional component, *remarks*, is any text. If used as the last of multiple statements on a single line, REM must be separated from the previous statement by a colon (:). The apostrophe (') synonym is similar to using REM, but no space is required between the apostrophe and the remarks, and no separating colon is needed on multi-statement lines.

Example

```
REM REM Example
REM This example does absolutely nothing
'It doesn't even have a single line of
executable code
```

Output

Related Items

REPLACE

Function

REPLACE(*target*, *find*, *source*[, *start*[, *count*[, *compare*]]])

Description

REPLACE returns a string which has had one substring replaced by another a specified number of times. The required parameter, *target*, is a string expression containing the substring to replace. The required parameter, *find*, is the string expression to be replaced. The required parameter, *source*, is the string expression used as the replacement. The optional parameter, *start*, is a numeric expression that specifies the starting position of the search within the target string; if *start* is not provided, the search defaults to 1, which represents the first character position. The optional parameter, *count*, is a numeric expression that specifies the number of substitutions to perform; if *count* is not provided -1 is used and all possible substitutions are made. The optional parameter, *compare*, is used to specify the type of search performed. See the Filter function for values.

Example

```
REM REPLACE Example
'REPLACE performs string substitutions
DIM Message, Ride
Message = "Good morning, class."
PRINT Message
Message = REPLACE(Message, "morning", _
    "afternoon")
PRINT Message
Ride = "big Al's big boat ride"
Ride = REPLACE(Ride, "big", "Big", 1, 1)
PRINT Ride
```

Output

```
Good morning, class.
Good afternoon, class.
Big Al's big boat ride
```

Related Items

RGB	Function
-----	----------

RGB(*red, green, blue*)

Description

RGB returns a whole number representing an RGB color value. *Red*, *Green* and *Blue* are numbers from 0-255 representing their respective components in the color.

Example

```
REM RGB Example
'RGB returns RGB Color
addObject "PictureBox", "PB", 50, 50, 20, 20
REM Turn color to red
PB.backcolor=rgb(255,0,0)
REM Turn color to grey
PB.backcolor=RGB(127,127,127)
REM Turn color to white
PB.backcolor=RGB(255,255,255)
```

Output

(a red rectangle, then a gray rectangle, then a white rectangle)

Related Items

RIGHT	Function
--------------	-----------------

RIGHT(*string*, *length*)
RIGHTB(*string*, *length*)

Description

RIGHT returns a string containing a number of characters from the right end of a string. The required parameter, *string*, is any valid string expression. The required parameter, *length*, is any valid numeric expression, if *length* is 0, an empty string ("") is returned, if *length* is greater than the size of string, the entire string is returned.

RIGHTB is a strict bitwise version of RIGHT. When used on the Unicode strings of Windows CE devices, it will return any partial character as a normal character.

Example

```
REM RIGHT Example
'RIGHT returns substring from string right end
DIM Wendy, Eric
Wendy = "Testaburger"
Eric = "Cartman"
PRINT "The RIGHT 6 of " & Wendy & ":", _
    RIGHT(Wendy, 6)
PRINT "The RIGHT 4 of " & Eric & ":", _
    RIGHT(Eric, 4)
```

Output

```
The RIGHT 6 of Testaburger:    burger
The RIGHT 4 of Cartman:       tman
```

Related Items

LEFT, MID

RND	Function
------------	-----------------

RND[(*number*)]

Description

RND returns a single-precision number from a pseudo-random sequence between 0 and 1. The optional parameter, *number*, is any valid numeric expression used to seed the random-number generator.

Table 22: RND seed values

Seed	RND generates
< 0	Same number every time
> 0 or None	Next random-number in sequence
0	Last number generated

Example

```
REM RND Example
'RND generates random numbers
Random
RND -1
Random
RND -1
Random

SUB Random
  DIM Ret, i
  RET = ""
  FOR i = 1 TO 4
    RET = RET & (INT(100 * RND) + 1) & " "
  NEXT
  PRINT "Four random numbers:", RET
END SUB
```

Output

```
Four random numbers:  6 71 1 19
Four random numbers: 33 28 4 51
Four random numbers: 33 28 4 51
```

Related Items

RANDOMIZE

ROUND**Function**

ROUND(*number*[, *fractionaldigits*])

Description

ROUND returns a number that has been rounded to the specified number of decimal places. The required argument, *number*, is any valid numeric expression. The optional argument, *fractionaldigits*, is the number of decimal places included in the rounding; if *fractionaldigits* is not provided it defaults to 0 and ROUND returns integers.

Example

```
REM ROUND Example
'ROUND rounds numbers to a given decimal place
DIM Pi, Pure, Ate
Pi = ROUND(3.14159265, 4)
PRINT Pi
Pure = ROUND(99.4444, 2)
PRINT Pure
Ate = ROUND(SQR(69))
PRINT Ate
```

Output

```
3.1416
99.44
8
```

Related Items

INT, FIX

RTRIM**Function**

RTRIM(*string*)

Description

RTRIM returns *string* with all trailing spaces removed. The required parameter, *string*, is any valid string expression.

Example

```
REM RTRIM Example
'RTRIM trims all trailing spaces
DIM Spacey
Spacey = "K-"
PRINT "(" & Spacey & ")"
PRINT "(" & RTRIM(Spacey) & ")"
```

Output

```
(K-)
(K)
```

Related Items

LTRIM, TRIM

RUNAPPATEVENT **Statement**

RUNAPPATEVENT *app, event*

Description

RUNAPPATEVENT is used to launch a program in response to a specific operating system event. The required component, *app*, is command line text to be executed. The required component, *event*, is the system event that causes the program to be launched.

Event	Value
NONE	0
TIME_CHANGE	1
SYNC_END	2
ON_AC_POWER	3
OFF_AC_POWER	4
NET_CONNECT	5
NET_DISCONNECT	6
DEVICE_CHANGE	7
IR_DISCOVERED	8
RS232_DETECTED	9
RESTORE_END	10
WAKEUP	11
TZ_CHANGE	12

Example

```
REM RUNAPPATEVENT Example
'RUNAPPATEVENT launches a program triggered
'by a system event
RUNAPPATEVENT "\Windows\player.exe", 3
```

Related Items

Runappatime

RUNAPPATTIME	Statement
---------------------	------------------

RUNAPPATTIME *app, yy, mo, dd, hh, mm, ss*

Description

RUNAPPATTIME is used to launch a program at a specified time in the future. The required component, *app*, is command line text to be executed. The required components, *yy, mo, dd, hh, mm*, and *ss*, are numeric values used to set the date and time the program will be launched.

Example

```
REM RUNAPPATTIME Example
'RUNAPPATTIME launches a program at a
specific time
RUNAPPATTIME "\Windows\player.exe", 2004, _
    5, 25, 14, 30
```

Related Items

RunappatEVENTT

ScrollBar	Object
------------------	---------------

ADDOBJECT "HScrollbar", *name*, *xpos*, *ypos*, *width*, *height*
ADDOBJECT "VScrollbar ", *name*, *xpos*, *ypos*, *width*, *height*

Description

Displays a scrollbar. The Change event is fired when the Right, Left, Up or Down arrows are tapped or when the slider is moved. SmallChange must be set to at least 1.

Properties Supported (see "Properties")

Enabled, Height, HWnd, LargeChange, Left, Max, Min, Name, ParentHWnd, SmallChange, TabStop, Tag, Top, Timer, Value, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Click, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timer

Example

```
REM VScrollbar Example
ADDOBJECT "VScrollbar", "Button", _
    120, 10, 20, 100
```

Output**Example**

```
REM HScrollbar Example
ADDOBJECT " HScrollbar ", "Button", _
    10, 120, 100, 20

SUB Button_Change
    PRINT "Button clicked"
    KillFocus
END SUB
```

Output**Related Items**

ADDOBJECT, Events, Methods, Properties

SECOND**Function**

SECOND(*time*)

Description

SECOND returns a whole number ranging from 0 to 59 that represents the second of the minute, of a given time. The required parameter, *time*, can be any numeric or string expression, or any expression that represents a time.

Example

```
REM SECOND Example
'SECOND returns second of minute of given time
PRINT "The SECOND of " & NOW & " is " _
      & SECOND(NOW)
```

Output

The SECOND of 8/18/1998 10:52:44 PM is 44

(sample date output is system dependant)

Related Items

DATE, DAY, HOUR, MINUTE, MONTH, NOW, TIME, YEAR

SELECT CASE	Statement
--------------------	------------------

```
SELECT CASE testexpression
    [CASE expressionlistA
        [statementsA]]
    [CASE expressionlistB
        [statementsB]]
    [CASE expressionlistC
        [statementsC]]...
    [CASE ELSE
        [elstatements]]
END SELECT
```

Description

SELECT CASE evaluates a test expression, to conditionally execute one of several groups of statements. An *expressionlistN* component is required for every optional CASE clause inside of SELECT CASE. The optional component, *statementsN* is the group of statements to be executed when *testexpression* matches any expression in a *expressionlistN*. Inside a CASE clause, statements are executed up to the next CASE clause or END SELECT; after all statements in a CASE clause are executed, execution continues with the next statement after END SELECT. If *testexpression* doesn't match any expression in any of the *expressionlists* and CASE ELSE is included, *elstatements* are executed. If CASE ELSE is not included and *testexpression* doesn't match any expression in any of the *expressionlists*, execution continues with the next statement after END SELECT.

Example

```
REM SELECT CASE Example
'SELECT CASE performs conditional execution
CheckHat("Blue")
CheckHat("Orange")
SUB CheckHat(Hat)
    SELECT CASE Hat
    CASE "Blue"
        PRINT "Kyle's hat"
    CASE "Green"
        PRINT "Stan's hat"
    CASE "Cyan"
        PRINT "Eric's hat"
    CASE "Orange", "Hood"
        PRINT "Kenny's hat"
    CASE "White"
        PRINT "Chef's hat"
    CASE "Striped"
        PRINT "Mr. Hat"
    CASE "Christmas", "Santa"
```

```
        PRINT "Mr. Hankey's hat"  
    CASE ELSE  
        PRINT "Unknown Hat"  
    END SELECT  
END SUB
```

Output

```
Kyle's hat  
Kenny's hat
```

Related Items

IF...THEN...ELSE

SENDKEY	Statement
----------------	------------------

SENDKEY *keyFlags*, *keyChar*

Description

SENDKEY is used to perform keyboard input programmatically. The required component, *keyFlags*, is a combination of modifier keys (shift, ctrl or alt) to accompany the key press. The required component, *keyChar*, is the numeric value of the key to be pressed.

Example

```
REM SENDKEY Example
'SENDKEY performs keyboard input
TextBox1.SetFocus
SENDKEY 0, 97
SENDKEY 0, 98
SENDKEY 0, 99
'Enters abc into the text box
```

SENDMESSAGE	Function
--------------------	-----------------

SENDMESSAGE (*hwnd, uMsg, wParam, lParam*)

SENDMESSAGESTRING (*hwnd, uMsg, wParam, lParam*)

Description

SENDMESSAGE passes directly through to the Windows API call of the same name. It's a very powerful function with all sorts of uses. Please consult Microsoft's API documentation for full information.

Example

```
SendMessage(obj.hWnd, WM_SETREDRAW, False, 0)
SendMessageString(Htm.hWnd, WM_SETTEXT, 0, "")
```

SET	Statement
------------	------------------

SET *objectvariable* = {*objectexpression* | NOTHING}

Description

SET is used to assign an object reference to a variable. The required component, *objectvariable*, is a variable that follows standard variable naming conventions. The required component, *objectexpression*, is the name of an object, a variable containing an object reference, or FUNCTION call that returns an object. The optional keyword, NOTHING, removes the association between *objectvariable* and any given object. When an object has no variables which reference it, the system and memory resources allocated to it are released.

Example

```
REM SET Example
'SET assigns objects to variables
DIM ButtonRef
ADDOBJECT "PictureBox", "Button", 0, 0, 10, 10
SET ButtonRef = Button
```

Output

Related Items

IS

SETMENU**Statement**

SETMENU "*menustring*[|*menukey*]", *menulist*

Description

SETMENU adds custom menus to the output window while a program is running. The required component, *menustring*, is the text that will be displayed in the menu item. The optional component, *menukey*, is the variable name that will be associated with the menu item. A *menukey* is separated from the *menustring* by two vertical bar characters(|), and should be used: to abbreviate a long *menustring*, when the *menustring* contains spaces or special characters, or to have multiple menu items with the same *menustring*. If no *menukey* is included, the value of *menustring* is used. The required component, *menulist*, is an array of string expressions that represent the *menukeys* of a submenu under a menu item. The *menukey* of the root menu of the output window is "Titlebar."

When a menu item is selected, an event is sent to the program. To respond to the event, include a PUBLIC SUB procedure with the following syntax:

```
SUB menukey_click()  
    'Do something in response  
END SUB
```

Any *menustring* that begins with a hyphen character (-) will be added to the menu as an unselectable separator. Separators do not support submenus.

SETMENU can be used to change menus dynamically. Any time SETMENU is called, the affected menus are updated in the output window. If a menu item is modified, all of its submenu items are cleared.

To aid keyboard menu navigation, a character in a *menustring* may be underlined. Insert an ampersand character (&) in the *menustring* before a character to have it underlined.

To display a second column in a menu (i.e. to show keyboard accelerators), append a tab character (vbTAB) and the desired second column string to the *menustring*.

Example

```
REM SETMENU Example  
'SETMENU adds custom menus to output window  
DIM Menu  
Menu = ARRAY("File", "Edit", "Help")  
SETMENU "Titlebar", Menu  
Menu = ARRAY("&New" & vbTAB & "Ctrl+n|New", _
```



```

"&Open" & vbTAB & "Ctrl+o||Open", _
"&Save" & vbTAB & "Ctrl+s||Save", _
"-", _
"&Recent Files||Recent", _
"-", _
"E&xit" & vbTAB & "Alt+F4||Exit")
SETMENU "File", Menu
Menu = ARRAY("&1 File 1||RF1", _
"&2 File 2||RF2", "&3 File 3||RF3", _
"&4 File 4||RF4")
SETMENU "Recent", Menu
SETMENU "Edit", ARRAY("Cut", "Copy", "Paste")
Menu = ARRAY("About SETMENU Example...||ASE")
SETMENU "Help", Menu
SUB New_Click()
'Open a new file
END SUB
SUB Open_Click()
'Open an existing file
END SUB

```

Output

SETPARENT	Statement
------------------	------------------

SETPARENT *child, parent*

Description

SETPARENT is used to move one control inside a container control. The required component, *child*, is the control to be moved into the container. The required component, *parent*, is the control that will hold the *child*, it must be a control with the container property (ie. Frame). Both controls must be created before the call to SETPARENT is made. This will work properly when the controls are intrinsic controls built into NS Basic. Third party controls do not all implement the needed interfaces to do this properly.

Example

```
REM SETPARENT Example
'SETPARENT moves one control inside another
ADDOBJECT "TextBox", "input", 5, 5, 140, 18
ADDOBJECT "Frame", "box", 0, 0, 200, 40
SETPARENT input, box
```

SGN**Function**

SGN(*number*)

Description

SGN returns an integer indicating the sign of a number. The required parameter, *number*, is any valid numeric expression. If *number* is less than zero -1 is returned, if *number* is greater than zero 1 is returned, if *number* is equal to zero, 0 is returned.

Example

```
REM SGN Example
'SGN returns the sign of a number as -1 or 1
DIM Pos, Neg, Zero
Pos = SGN(44)
Neg = SGN(-17)
Zero = SGN(100 - 100)
PRINT "Positive:", Pos
PRINT "Negative:", Neg
PRINT "Zero:", Zero
```

Output

```
Positive:      1
Negative:     -1
Zero:         0
```

Related Items

ABS

SHELLEXECUTE**Statement**

SHELLEXECUTE *verb*, *file* [,*parms*]

Description

SHELLEXECUTE executes an external program as a separate process. The first parameter, *verb*, tells what action should be taken with the *file* in the second parameter. The third parameter contains any parameters that are required.

Table 23:

Verb	What it does
Open	Runs a program
Print	Sends a document to the printer (not available on all Windows CE devices)
Explore	Opens Pocket Internet Explorer

Example

```
REM SHELLEXECUTE Example
'Register a third party control
shellExecute "open","regsvrce.exe",_
"\windows\s309picture.dll"
'print a document (not on all devices)
ShellExecute "print","myReport.doc"
'open a picture using web browser
shellExecute "explore", "myPict.jpg"
```

Output

(varies)

Related Items

SHOWFULLSCREEN	Statement
-----------------------	------------------

SHOWFULLSCREEN *TRUE|FALSE*

Description

SHOWFULLSCREEN is used to show and hide various sections of the screen that are reserved by the operating system. This statement only works on Windows Mobile and Pocket PC devices.

Example

```
REM SHOWFULLSCREEN show/hide areas of the
screen
SHOWFULLSCREEN TRUE      'Shows full screen
```

SHOWOKBUTTON	Statement
---------------------	------------------

SHOWOKBUTTON *true|false*

Description

SHOWOKBUTTON is used to show or hide the OK button in the menubar of the output window of Pocket PC devices. By default the OK button is hidden and the close button (X) is visible. Tapping the close button causes the output window to minimize and continue execution. When the OK button is visible and it is tapped, the output window is closed and the program exits.

Example

```
REM SHOWOKBUTTON Example
'SHOWOKBUTTON shows or hides OK in menu bar
SHOWOKBUTTON true
```

SIN**Function**

SIN(*number*)

Description

SIN calculates the sine of a number expressing an angle in radians. The required parameter, *number*, is any numeric expression. The value returned is a double-precision floating-point number that can range from -1 to 1.

To convert degrees to radians, multiply degrees by $\pi/180$.
To convert radians to degrees, multiply radians by $180/\pi$.

Example

```
REM SIN Example
'SIN calculates the sine of a number
PRINT "The sine of 0 is " & SIN(0)
```

Output

The sine of 0 is 0

Related Items

COS, TAN

SLEEP	Statement
--------------	------------------

SLEEP *number*

Description

SLEEP yields the CPU for other processes. The required parameter, *number*, is a numeric expression that specifies the number of milliseconds to sleep.

Example

```
REM SLEEP Example
'SLEEP yields the CPU for other processes
PRINT NOW
SLEEP(5000)
PRINT NOW
```

Output

```
8/18/1998 10:52:44 PM
8/18/1998 10:52:49 PM
```

Related Items

SPACE**Function**

SPACE(*number*)

Description

SPACE returns a string consisting of a number of spaces. The required argument, *number*, is any valid numeric expression.

Example

```
REM SPACE Example
'SPACE creates a string of spaces
DIM Spaces, Letter
Spaces = SPACE(4)
FOR i = 0 to 4
    Letter = LEFT(Spaces, i) & CHR(65 + i)
    PRINT Letter
NEXT
```

Output

```
A
 B
  C
   D
    E
```

Related Items

STRING

SPECIALFOLDER	Function
---------------	----------

SPECIALFOLDER(*ID* [, *True*])

Description

Get special folder names. If *True* is added, the folder will be created if it does not exist. This function allows you to use the proper folder names at runtime on devices in languages other than your own, and to prepare for future devices. This function is supported on Windows CE 3.0 and later devices. We have also found the output from this function is device dependant.

2	\\Windows\\Start Menu\\Programs
3	The file system directory that contains the user's program groups, which are also file system directories.
5	The file system directory that serves as a common repository for documents.
6	The file system directory that serves as a common repository for the user's favorite items.
7	The file system directory that corresponds to the user's Startup program group. The system starts these programs when a device is powered on.
8	File system directory that contains the user's most recently used documents.
13	Folder that contains music files.
15	Folder that contains video files.
16	File system directory used to physically store file objects on the desktop (not to be confused with the desktop folder itself).
20	The virtual folder that contains fonts.
26	File system directory that serves as a common repository for application-specific data.
36	The Windows folder.
38	The program files folder.
39	Folder that contains picture files.
40	Folder that contains the profile of the user.

Example

```
MsgBox SpecialFolder(6)
```

Output

(displays "\\Windows\\Favorites" on English Pocket PC)

Related Items

SPLIT**Function**

SPLIT(*string*[, *delimiter*[, *count*[, *compare*]]])

Description

SPLIT returns a one-dimensional array of strings of a specified length, created by dividing a single string where a specified delimiter occurs. The required parameter, *string*, is a valid string expression; if *string* is a zero-length string (""), SPLIT returns an empty array. The optional argument, *delimiter*, is a string expression whose first character is used to separate the substrings; if *delimiter* is a zero-length string (""), a single-element array containing the entire string is returned. The optional parameter, *count*, is a numeric expression that specifies the number of substrings to return; if *count* is not provided -1 is used as a default specifying that all substrings be returned. The optional parameter, *compare*, is used to specify the type of search performed.

Example

```
REM SPLIT Example
'SPLIT divides a string into substrings
DIM List, Who, All, TopTwo
List = "Eric,Kenny,Kyle,Stan"
Who = SPLIT(List, ",")
PRINT Who(0), Who(1), Who(2), Who(3)
All = "First Second Third Fourth Fifth"
TopTwo = SPLIT(All, " ")
PRINT TopTwo(0), TopTwo(1)
```

Output

```
Eric           Kenny   Kyle           Stan
First  Second
```

Related Items

JOIN

SQR**Function**

SQR(*number*)

Description

SQR returns a double-precision value representing the square root of a number. The required parameter, *number*, is any valid numeric expression greater than or equal to zero.

Example

```
REM SQR Example
'SQR calculates square root of a number
PRINT "The square root of 69 is " & SQR(69)
```

Output

The square root of 69 is 8.30662386291807

Related Items

STRCOMP**Function**

STRCOMP(*string1*, *string2*[, *compare*])

Description

STRCOMP compares two strings and returns an integer value which indicates the alphabetical relationship between them. The required parameters, *string1* and *string2*, are two valid string expressions. The optional parameter, *compare*, is used to specify the type of comparison performed. STRCOMP returns -1 if *string1* is less than *string2*, 0 if *string1* is equal to *string2*, or 1 if *string1* is greater than *string2*.

Example

```
REM STRCOMP Example
'STRCOMP compares two strings
Sort "Kenny", "Kyle", vbBinaryCompare
Sort "Eric", "eric", vbTextCompare
Sort "Wendy", "Stan", vbBinaryCompare
SUB Sort(string1, string2, compare)
    DIM Order
    Order = STRCOMP(string1, string2, compare)
    IF Order < 0 THEN
        PRINT string1 & " precedes " & string2
    ELSEIF Order > 0 THEN
        PRINT string2 & " precedes " & string1
    ELSE
        PRINT string1 & " and " & string2 _
            & " are equivalent"
    END IF
END SUB
```

Output

```
Kenny precedes Kyle
Eric and eric are equivalent
Stan precedes Wendy
```

Related Items

STRING**Function**

STRING(*number*, *character*)

Description

STRING returns a string created by concatenating a given number of a specified character. The required argument, *number*, is any valid numeric expression. The required argument, *character*, is a character or string expression whose first character is repeated in the returned string.

Example

```
REM STRING Example
'String creates a string repeating a character
DIM Message
Message = STRING(10, "Hello World!")
PRINT Message
Message = STRING(10, CHR(ASC("i")))
PRINT Message
```

Output

```
HHHHHHHHHH
iiiiiiiiiii
```

Related Items

SPACE

STRREVERSE	Function
-------------------	-----------------

STRREVERSE(*string*)

Description

STRREVERSE returns a string created by reversing the character order of another string. The required argument, *string*, is any valid string expression.

Example

```
REM STRREVERSE Example
'STRREVERSE reverses characters in a string
DIM Show
Show = "Terrence and Phillip"
PRINT "Forwards: " & Show
PRINT "Reverse: " & STRREVERSE(Show)
```

Output

```
Forwards: Terrence and Phillip
Reverse: pillihP dna ecnerreT
```

Related Items

SUB	Statement
------------	------------------

```
SUB procedurename[(arglist)]  
    [statements]  
    [EXIT SUB]  
    [statements]  
END SUB
```

Description

SUB declares a procedure, *procedurename*, that executes *statements*, with *arglist* as parameters, with no return value. The required parameter, *procedurename*, is used to call the procedure, and must follow standard variable naming conventions. The optional parameter list, *arglist*, is a comma separated list of variables that are passed to the procedure when it is called. The optional component, *statements*, will be executed as the body of the procedure. Any number of optional EXIT SUB statements can be used to exit the procedure.

Each member of *arglist* is an argument passed to the procedure, as declared below:

```
[BYVAL | BYREF] varname[()]
```

The optional keyword, BYVAL, specifies that a copy of the variable be passed into the procedure, making its value outside the procedure unmodifiable by the procedure. The optional keyword, BYREF, specifies that the address of the variable be passed into the procedure, making its value outside the procedure modifiable by the procedure. By default, parameters are passed in BYREF.

A FUNCTION procedure will be called as a SUB procedure if the return value is not stored in a variable or used in an expression.

A SUB procedure called with zero or one arguments may be called with empty parenthesis or the single argument in parenthesis. NS Basic/CE treats a single expression enclosed in parenthesis as a single expression, not a one-element argument list.

Example

```
REM SUB Example  
'SUB declares a procedure  
DIM PriceA, PriceB  
PrintMenu "Wednesday"  
PriceA = 53  
PriceB = 44  
Sort PriceA, PriceB  
PRINT "Lowest Price:", PriceA  
SUB PrintMenu(day)  
    IF day = "Wednesday" THEN
```



```

        PRINT "Wednesday is Salisbury Steak day"
    END IF
END SUB
SUB Sort (BYREF x, BYREF y)
    DIM Temp
    IF x > y THEN
        Temp = y
        y = x
        x = Temp
    EXIT SUB
    ELSE
        Temp = x
        x = y
        y = Temp
    END IF
END SUB

```

Output

```

Wednesday is Salisbury Steak day
Lowest Price:  44

```

Related Items

CALL, FUNCTION

SYSINFO Function

SYSINFO(*number*)

Description

SYSINFO returns information about the device. This is not a complete list of return values: see Microsoft's documentation on `GetSystemMetrics` and `GetDeviceCaps` for more.

SM_CXSCREEN	0	Width of screen
SM_CYSCREEN	1	Height of screen
SM_CXVSCROLL	2	Width of arrow bitmap on vertical scroll bar
SM_CXHSCROLL	3	Height of arrow bitmap on horizontal scroll bar
SM_CYCAPTION	4	Height of caption or title
SM_CXDLGFRAME	7	Width of dialog frame window
SM_CYDLGFRAME	8	Height of dialog frame window
SM_CYMENU	15	Height of menu
SM_CXFULLSCREEN	16	Width of window client area
SM_CYFULLSCREEN	17	Height of window client area
SM_CYVSCROLL	20	Height of arrow bitmap on horizontal scroll bar
SM_CXVSCROLL	21	Height of arrow bitmap on vertical scroll bar
HORZREZ	108	HiRes Display width pixels
VERTREZ	109	HiRes Display height pixels
BITSPIXEL	112	Bits per pixel
NUMCOLORS	124	Number of colors
LOGPIXELSX	188	Pixels per inch horizontal
LOGPIXELSY	190	Pixels per inch vertical

Example

```
REM SYSINFO Example
MSGBOX "The device's DPI is " & sysInfo(188)
```

Output

The device's DPI is 96.

Related Items

TAN**Function**

TAN(*number*)

Description

TAN calculates the tangent of a number expressing an angle in radians. The required parameter, *number*, is any numeric expression. The value returned is a double-precision floating-point number.

To convert degrees to radians, multiply degrees by $\pi/180$.
To convert radians to degrees, multiply radians by $180/\pi$.

Example

```
REM TAN Example
'TAN calculates the tangent of a number
PRINT "The tangent of 0 is " & TAN(0)
```

Output

The tangent of 0 is 0

Related Items

COS, SIN

TextBox**Object**

ADDOBJECT "TextBox", *name*, *xpos*, *ypos*, *width*, *height*

Description

TextBox is used to display editable text in an object that floats over the output window. A TextBox allows text input from the keyboard. The required component, *name*, is the name of a variable that is added to the program to reference the object, it must follow standard variable naming conventions. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner. **Note:** Set the Multiline property to TRUE to allow embedded carriage returns and wrapped text.

Properties Supported (see "Properties")

BackColor, BorderStyle, Caption, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, ForeColor, Height, HideSelection, Hwnd, Left, Locked, LowercaseOnly, MaxLength, MultiLine, Name, NumbersOnly, ParentHwnd, Password, Scrollbars, selLength, selStart, selText, TabStop, Tag, Text, Timer, Top, UppercaseOnly, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Change, Click, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, Timer

Example

```
REM TextBox Example
'TextBox is an editable text field
ADDOBJECT "TextBox", "Text", 10, 25, 90, 90
Text.Text = "Hello World!"
SUB Text_Click
    PRINT "TextBox Text: " & Text.Text
END SUB
```

Output**Related Items**

ADDOBJECT

TIME	Function
-------------	-----------------

TIME

Description

TIME returns the current system time.

Example

```
REM TIME Example
'TIME returns current system time
DIM RightNow
RightNow = TIME
PRINT "The time now is " & RightNow
```

Output

The time now is 10:52:44 PM

(sample time output is system dependant)

Related Items

DATE, NOW, DATEADD (and many more)

Time**Object**

ADDOBJECT "Time", *name*, *xpos*, *ypos*, *width*, *height*

Description

Time is used to display a standard time picker object in the output window. The required component, *name*, is the name of a variable that is added to the program to reference the object. The required components, *xpos*, *ypos*, *width*, and *height* are numeric expressions that set the location and size of the object in pixels, measured from the upper left corner of the output window. Do not use a MSGBOX inside your Time_Change event: it will cause an error. This object is not available on Windows CE 2.0 devices. Use the .Date property to set the time.

Properties Supported (see "Properties")

BorderStyle, Date, Enabled, FontBold, FontItalic, FontName, FontSize, FontStrikethru, FontUnderline, Height, Hwnd, Left, Name, ParentHwnd, TabStop, Tag, Text, Timer, Top, Visible, Width, WindowLong

Methods Supported (see "Methods")

Hide, Move, SetFocus, Show

Events Supported (see "Events")

Change, DropDown

Example

```
REM Time Example
ADDOBJECT "Time", "Time", 100,100,120,20
Time.Date = "09/02/04 03:57:01 PM"
SUB Time_Change
    PRINT "Time Changed to " & time.text
END SUB
```

Output**Related Items**

ADDOBJECT, Events, Methods, Properties

<u>TIMESERIAL</u>	<u>Function</u>
--------------------------	------------------------

TIMESERIAL(*hour, minute, second*)

Description

TIMESERIAL returns a time constructed from the given hour, minute, and second. The required parameter, *hour*, is any numeric expression ranging from 0 to 23. The required parameters, *minute* and *second*, can be any numeric expression.

Example

```
REM TIMESERIAL Example
'TIMESERIAL builds a time from its parts
DIM FiveThirty, Noon
FiveThirty = TIMESERIAL(11 - 6, 30, 0)
Noon = TIMESERIAL(12, 0, 0)
PRINT "Half past five:", FiveThirty
PRINT "Noon:", Noon
```

Output

```
Half past five:      05:30:00 AM
Noon:  12:00:00 PM
```

(sample time output is system dependant)

Related Items

DATESERIAL

TIMEVALUE	Function
------------------	-----------------

TIMEVALUE(*time*)

Description

TIMEVALUE returns a time from, *time*, which is usually a string, but any expression that can represent a time ranging from 0:00:00 (12:00:00 AM) to 23:59:59 (11:59:59 PM), can be used.

If *time* is a string that consists of numbers separated by time separators, it recognizes hour, minute, and second in the order specified by the system short time format, using zero for unspecified components. TIMEVALUE recognizes time in both 12-hour and 24-hour format.

Time separators are characters that separate hour, minute and second when a time is formatted as a string, they are determined by your system settings.

Example

```
REM TIMEVALUE Example
'TIMEVALUE returns a time
DIM FiveThirty, Noon
FiveThirty = TIMEVALUE("5:30 PM")
Noon = TIMEVALUE("12:00")
PRINT "Half past five:", FiveThirty
PRINT "Noon:", Noon
```

Output

```
Half past five:      05:30:00 PM
Noon:  12:00:00 PM
```

(sample time output is system dependant)

Related Items

DATEVALUE

TRIM	Function
-------------	-----------------

TRIM(*string*)

Description

TRIM returns a string with all leading and trailing spaces removed. The required parameter, *string*, is any valid string expression.

Example

```
REM TRIM Example
'TRIM trims leading and trailing spaces
DIM Spacey
Spacey = "-K-"
PRINT "(" & Spacey & ")"
PRINT "(" & TRIM(Spacey) & ")"
```

Output

```
(-K-)
(K)
```

Related Items

LTRIM, RTRIM

TYPENAME	Function
----------	----------

TYPENAME(*variable*)

Description

TYPENAME returns a string specifying the type of a variable. The required parameter, *variable*, is any variable.

Table 24: TYPENAME return values

Return value	Description
Boolean	Boolean value
Byte	Byte value
Currency	Currency value
Date	Date value
Decimal	Decimal value
Double	Double-precision floating-point value
Empty	Uninitialized
Error	An error value
Integer	Integer value
Long	Long integer value
Nothing	Object variable that doesn't contain an object reference
Null	No valid value
Object	Generic object
Oobjecttype	An object of type oobjecttype
Single	Single-precision floating-point value
String	String value
Unknown	Unknown
Variant()	Array

Example

```
REM TYPENAME Example
'TYPENAME returns variable type as a string
DIM nInteger, nSingle
nInteger = CINT(44)
PRINT 44 & " is a/an " & TYPENAME(nInteger)
nSingle = CSNG(99.44)
PRINT 99.44 & " is a/an " &
TYPENAME(nSingle)
```

Output

44 is a/an Integer

99.44 is a/an Single

Related Items

ISARRAY, ISDATE, ISEMPY, ISNULL, ISNUMERIC,
ISOBJECT, VARTYPE

UBOUND	Function
---------------	-----------------

UBOUND(*array* [, *dimension*])

Description

UBOUND returns a long value specifying the largest available subscript in the specified dimension of a given array. The required parameter, *array*, is any array variable. The optional parameter, *dimension*, specifies which dimension's upper bound is returned, beginning with the default, 1.

Example

```
REM UBOUND Example
'UBOUND returns upper bound of array dimension
DIM Other, Children(3), Parents(3, 1)
Other = ARRAY("Damien", "Pip", "Wendy")
PRINT "'Other' Upper Bound:", UBOUND(Other)
PRINT "'Children' Upper Bound:", _
      UBOUND(CHILDREN, 1)
PRINT "'Parents' Upper Bounds:", _
      UBOUND(Parents), UBOUND(Parents, 2)
```

Output

```
'Other' Upper Bound:  2
'Children' Upper Bound:      3
'Parents' Upper Bounds:      3      1
```

Related Items

ARRAY, DIM, LBOUND, REDIM

UCASE	Function
--------------	-----------------

UCASE(*string*)

Description

UCASE returns string with all of its lowercase characters converted to uppercase. The required parameter, *string*, is any valid string expression.

Example

```
REM UCASE Example
'UCASE returns string with all uppercase chars
DIM Vet
Vet = "ned"
PRINT Vet & " uppercase is " & UCASE(Vet)
```

Output

```
ned uppercase is NED
```

Related Items

LCASE

UPDATESCREEN	Statement
UPDATESCREEN	
Description	
UPDATESCREEN forces the output window to redraw.	
Example	
<pre> REM UPDATESCREEN Example 'UPDATESCREEN redraws the output window UPDATESCREEN </pre>	
Output	
Related Items	

VARTYPE**Function**

VARTYPE(*variable*)

Description

VARTYPE returns an integer that indicates the type of a variable. The required parameter, *variable*, is any variable that doesn't contain a user-defined type.

When *variable* is an array, the value returned is equal to the array constant plus the constant that specifies the element-type.

Table 25: VARTYPE return values

Constant	Value	Description
vbEmpty	0	Uninitialized (default)
vbNull	1	No valid data
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point
vbDouble	5	Double-precision floating-point
vbCurrency	6	Currency
vbDate	7	Date
vbString	8	String
vbObject	9	Object
vbError	10	Error
vbBoolean	11	Boolean
vbVariant	12	Variant (only with arrays of Variants)
vbDataObject	13	Data-access object
vbByte	17	Byte
vbArray	8192	Array

Example

```
REM VARTYPE Example
'VARTYPE returns variable type as an integer
DIM nInteger, nSingle
nInteger = CINT(44)
PRINT 44 & " is VARTYPE " &
VARTYPE(nInteger)
nSingle = CSNG(99.44)
PRINT 99.44 & " is VARTYPE " &
VARTYPE(nSingle)
```

Output

```
44 is VARTYPE 2  
99.44 is VARTYPE 4
```

Related Items

ISARRAY, ISDATE, ISEMPY, ISNULL, ISNUMERIC,
ISOBJECT, TYPENAME

WAITCURSOR	Statement
-------------------	------------------

WAITCURSOR true|false

Description

Set WAITCURSOR to true to display a wait cursor. The design of the cursor depends on the device. Set it to false to turn the waitcursor off again.

Example

```
REM WAITCURSOR Example
WAITCURSOR TRUE
SLEEP 5000
WAITCURSOR FALSE
```

Output

Related Items

WAVEVOLUME	Function
-------------------	-----------------

WAVEVOLUME *volume*

Description

WAVEVOLUME is used to set the main volume level of the device. The required component, *volume*, is a value between 0 and 65535.

Example

```
REM WAVEVOLUME Example
'WAVEVOLUME sets the volume level of the
device
WAVEVOLUME=0           'Mute on
WAVEVOLUME=32768       'Volume 50%
WAVEVOLUME=65535       'Full volume
MSGBOX WAVEVOLUME
```

Output

(MsgBox with 65536)

Related Items

PLAYSOUND

WEEKDAY**Function**

WEEKDAY(*date*[, *firstdayofweek*])

Description

WEEKDAY returns an integer representing the day of week from a given date. The required parameter, *date*, is a numeric expression, a string expression, or any valid expression that can represent a date. The optional parameter, *firstdayofweek*, is Sunday, if not specified. The return value is an integer from the table below.

For more information about *firstdayofweek* constants, see "DateDiff".

Table 26: WEEKDAY return values

Constant	Value	Description
vbSunday	1	Sunday
vbMonday	2	Monday
vbTuesday	3	Tuesday
vbWednesday	4	Wednesday
vbThursday	5	Thursday
vbFriday	6	Friday
vbSaturday	7	Saturday

Example

```
REM WEEKDAY Example
'WEEKDAY returns day of week as an integer
DIM IndepDay, Birthday
IndepDay = WEEKDAY("July 4, 1776")
PRINT "WEEKDAY of July 4, 1776:", IndepDay
Birthday = WEEKDAY("12/27/70")
PRINT "WEEKDAY of 12/27/70:", Birthday
```

Output

```
WEEKDAY of July 4, 1776:      5
WEEKDAY of 12/27/70:      1
```

Related Items

WEEKDAYNAME	Function
--------------------	-----------------

WEEKDAYNAME(*number*[, *abbreviate*[, *firstdayofweek*]])

Description

WEEKDAYNAME returns a string representing the day of week. The required argument, *number*, is an integer ranging from 1 to 7 that represents the number of the day of the week. The optional argument, *abbreviate*, is a boolean expression, if *abbreviate* is TRUE the value returned is the three-letter abbreviation of the weekday name, otherwise the full name is returned by default. The optional parameter *firstdayofweek* is Sunday, if not specified.

Example

```
REM WEEKDAYNAME Example
'WEEKDAYNAME returns string name of day
DIM Day1, Day2
Day1 = 1
PRINT WEEKDAYNAME(Day1)
Day2 = 2
PRINT WEEKDAYNAME(Day2, TRUE, vbMonday)
```

Output

```
Sunday
Tue
```

Related Items

WITH..END WITH	Statement
-----------------------	------------------

```
WITH object  
    [statements]  
END WITH
```

Description

WITH allows you to do a series of operations on an object without having to name the object each time. Windows CE 4.0 and later only.

Example

```
AddObject "CommandButton", "CB", 124, 60, 108, 21  
With CB  
    .FontBold = True  
    .Caption = "Tap Me!"  
    .FontItalic = True.  
End With
```

Output

(CommandButton that says "Tap ME!" in bold italics)

Related Items

CLASS

WHILE...WEND	Statement
---------------------	------------------

```
WHILE condition
    [statements]
WEND
```

Description

WHILE...WEND repeats a group of statements while a given condition is TRUE. The required component, *condition*, is any valid expression that evaluates to TRUE or FALSE. The optional component, *statements*, are executed during each iteration of the loop. WHILE...WEND statements can be nested, and any WEND statements in a nested loop transfer execution to one level above the loop where the WEND occurs.

Example

```
REM WHILE...WEND Example
'WHILE...WEND repeats a group of statements
DIM Counter
Counter = 1
WHILE Counter < 5
    PRINT "Counter = " & Counter
    Counter = Counter + 1
WEND
```

Output

```
Counter = 1
Counter = 2
Counter = 3
Counter = 4
```

Related Items

DO...LOOP, FOR...NEXT, FOR EACH...NEXT

result = *x* XOR *y*

Description

XOR returns the logical, exclusive disjunction of two expressions. *result* is TRUE, if and only if one of the expressions *x* and *y* evaluate to TRUE, otherwise, *result* is FALSE.

XOR also does a bitwise comparison of two numeric expressions. Each bit in *result* is set to 1 if and only if one of the corresponding bits in *x* or *y* is 1, otherwise it is set to 0.

Example

```
REM XOR Example
'XOR performs exclusive disjunctions
DIM Test1, Test2, x, y
x = 2
y = 9
Test1 = x > 0 XOR y < 10
Test2 = x > 0 XOR y > 10
PRINT "Logical:"
PRINT "  x > 0 XOR y < 10 = " & CSTR(Test1)
PRINT "  x > 0 XOR y > 10 = " & CSTR(Test2)
PRINT "Bitwise:"
PRINT "  x XOR y = " & (x XOR y)
```

Output

```
Logical:
  x > 0 XOR y < 10 = False
  x > 0 XOR y > 10 = True
Bitwise:
  x XOR y = 11
```

Related Items

AND, EQV, IMP, NOT, OR

YEAR	Function
-------------	-----------------

YEAR(*date*)

Description

YEAR returns an integer, that represents the year of the given date. The required parameter *date* can be any expression that represents a date.

Example

```
REM YEAR Example
'YEAR returns year of a date as an integer
DIM IndepYear, Birthyear
IndepYear = YEAR("July 4, 1776")
PRINT "YEAR of July 4, 1776:", IndepYear
Birthyear = YEAR("12/27/70")
PRINT "YEAR of 12/27/70:", Birthyear
```

Output

```
YEAR of July 4, 1776: 1776
YEAR of 12/27/70:    1970
```

Related Items

DAY, HOUR, MINUTE, MONTH, NOW, SECOND, TIME

6. Advanced Topics

6.1 Coding Conventions

Coding conventions are a collection of suggestions that can aid in the creation, reading, debugging, and maintaining of programs. Coding conventions include, but are not limited to:

- Naming guidelines
- Text formatting guidelines
- Commenting guidelines

Solid coding conventions should help to provide context-sensitive information and improved visual readability.

6.1.1 Naming guidelines

The easiest way to provide context-sensitive information in a program is by the use of concise, informative names of constants, variable, and procedures.

Non-trivial names should be as complete as necessary to describe purpose or use. When multiple words are used, vary upper- and lower-case letters and/or use an underscore (_) character to delimit words.

Constant names should be all uppercase with an underscore between words.

Variable names should be lower-case with the first letter of each word capitalized; refrain from using underscores in variable names. For additional information, start each variable name with a three letter prefix that describes the type of information that the variable will store (bln for BOOLEAN, int for INTEGER, str for STRING, etc.).

Procedure names should be lower-case with the first letter of each word capitalized. The use of underscores and mixed case will easily differentiate procedure names from constant or variable names.

6.1.2 Text formatting guidelines

Text formatting is an excellent way to improve the visual readability of a program. Use white vertically to space to group or separate statements, and horizontally to align statements to show logic structure and nesting.

Insert a blank line after constant definitions, after variable definitions, and after each procedure declaration to keep sections separated. A blank line can also be inserted after a related statements at script level or inside procedures; this will help to form groups.

The standard tab indent is two spaces. Use at least one tab to show nested groups of statements inside procedures, loops, and conditional execution statements.

6.1.3 Comment guidelines

Comments should document what the programmer is attempting to do and how, as well as what should be done or what needs to be done. Use comments:

- At the top of each program to include information about who wrote it, what services it can provide, when it was written, when updates occurred and what prompted them, and who owns the rights to the program.
- To preface each procedure with an explanation of the purpose of the procedure along with its expected inputs and any output. An additional line should be included for each variable that is being passed in if its purpose is not obvious or it has range restrictions.
- To append a brief statement after constant and variable declarations, adding information about scope and usage.
- Above a group of related statements to explain what the statements are doing.
- In conditional execution statements to further clarify each condition and how it comes about.

6.2 Handling Errors

In a perfect world, there are never any errors. Our programs are seldom perfect worlds! We protect our users (often ourselves) from many errors using two techniques: defensive programming and error trapping.

The basic idea with error handling is to anticipate which parts of your program could have run-time errors, and to set up special program code to deal with it.

6.2.1 Defensive programming

Programming defensively helps to avoid run-time errors by reducing the number of assumptions that are made as the program executes. In addition, you should always do the following:

- Initialize variables after they are declared to ensure they hold the proper type of data before they are used.

- Use the OPTION EXPLICIT statement to ensure that all variables are declared before being used. This helps to prevent mistakes from misspelling variable names.
- Use the VARTYPE function to ensure a variable whose value is input by the user contains the expected data type.
- Be sure that numeric values are within the valid range before using them in numeric expressions.
- Know the number of elements in an array dimension before trying to access them.

The more defensively you program, the less chance there is for run-time errors to occur. A good defensive programmer can eliminate logic errors and protect the program from any errors caused by user interaction.

6.2.2 Error trapping

Defensive programming can and should be used to prevent as many run-time errors as possible. When defensive programming can not protect your program from errors, the ON ERROR statement can be used to intercept errors that occur. This method is known as error trapping, and it allows you to process errors before NS Basic/CE gets notified that they have occurred.

Run-time errors that can not be prevented by defensive programming generally occur when a program is interacting with the operating system. File operations cause the most common errors, because they request system resources that may or may not be available at any given time.

NS Basic/CE only allows error trapping within procedures, and error trapping can easily be enabled or disabled on a procedure by procedure basis. When a program traps an error, it deals with it through an error handler. An error handler is simply a block of statements that gets executed to allow a program to deal with an error. Let's look at simple error trapping and handling with file operations:

6.2.3 Error trapping file operations

```
ADDOBJECT "File", "MyFile"
DIM FileOpen

FileOpen = Open("Some non-existent filename")
IF FileOpen THEN
    PRINT "File opened successfully."
ELSE
    PRINT "Unable to open file."
END IF

FUNCTION Open(filename)
    ON ERROR RESUME NEXT
    MyFile.Open filename, 1
```

```
IF ERR <> 0 THEN
    'This is the error handler that is used
    'when a file cannot be opened for reading.
    'This simple example just returns FALSE.
    Open = FALSE
    ERR.Clear
    EXIT FUNCTION
END IF
MyFile.Close
Open = TRUE
END FUNCTION
```

Output

Unable to open file.

Some actions your error handler may perform:

- Display a helpful message and retry the operation.
- Correct the error by setting one or more variables to some default value (i.e., you can limit an input to a maximum value).
- Display your own error message, perform some clean up (perhaps update a file), and then end the program.

A P P E N D I X

A

A. Error Codes

5	Invalid procedure call or argument
6	Overflow
7	Out of memory
9	Subscript out of range
10	Array fixed or temporarily locked
11	Division by zero
13	Type mismatch
14	Out of string space
17	Can't perform requested operation
28	Out of stack space
35	Sub or Function not defined
48	Error in loading DLL
51	Internal error
52	Bad file name or number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
61	Disk full
62	Input past end of file
67	Too many files
68	Device unavailable
70	Permission denied
71	Disk not ready
74	Can't rename with different drive
75	Path/File access error
76	Path not found
91	Object variable not set
92	For loop not initialized
94	Invalid use of Null
322	Can't create necessary temporary file
424	Object required
429	ActiveX component can't create object
430	Class doesn't support Automation
432	File name or class name not found during Automation operation
438	Object doesn't support this property or method
440	Automation error
445	Object doesn't support this action

446	Object doesn't support named arguments
447	Object doesn't support current locale setting
448	Named argument not found
449	Argument not optional
450	Wrong number of arguments or invalid property assignment
451	Object not a collection
453	Specified DLL function not found
455	Code resource lock error
457	This key already associated with an element of this collection
458	Variable uses an Automation type not supported in VBScript
500	Variable is undefined
501	Illegal assignment
502	Object not safe for scripting
503	Object not safe for initializing
504	Object not safe for creating
505	Invalid or unqualified reference
506	Class not defined
507	An exception occurred
32811	Element not found
32812	The specified date is not available in the current locale's calendar

A P P E N D I X

B

B. Constants

Color	vbBLACK, vbRED, vbGREEN, vbBLUE, vbYELLOW, vbBLUE, vbMAGENTA, vbCYAN, vbWHITE
Comparison	vbBINARYCOMPARE, vbTEXTCOMPARE
Date/Time	vbSUNDAY, vbMONDAY, vbTUESDAY, vbWEDNESDAY, vbTHURSDAY, vbFRIDAY, vbSATURDAY, vbFIRSTJAN1, vbFIRSTFOURDAYS, vbFIRSTFULLWEEK, vbUSESYSTEM, vbUSESYSTEMDAYOFWEEK
Date Format	vbGENERALDATE, vbLONGDATE, vbSHORTDATE, vbLONGTIME, vbSHORTTIME
MSGBOX	<p>Display options (add to combine): vbOKONLY, vbOKCANCEL, vbABORTRETRYIGNORE, vbYESNOCANCEL, vbYESNO, vbRETRYCANCEL, vbCRITICAL, vbQUESTION, vbEXCLAMATION, vbINFORMATION, vbDEFAULTBUTTON1, vbDEFAULTBUTTON2, vbDEFAULTBUTTON3, vbDEFAULTBUTTON4, vbAPPLICATIONMODAL, vbSYSTEMMODAL</p> <p>Return values: vbOK, vbCANCEL, vbABORT, vbRETRY, vbIGNORE, vbYES, vbNO</p>
String	vbCR, vbCRLF, vbFORMFEED, vbLF, vbNEWLINE, vbNULLCHAR, vbTAB, vbNULLSTRING, vbVERTICALTAB
VARTYPE	vbEMPTY, vbNULL, vbINTEGER, vbLONG, vbSINGLE, vbDOUBLE, vbCURRENCY, vbDATE, vbSTRING, vbOBJECT, vbERROR, vbBOOLEAN, vbVARIANT, vbDATAOBJECT, vbDECIMAL, vbBYTE, vbARRAY
Global	CurrentPath, NSBVersion, AppBuildStamp, AppComments, AppCompanyName, AppFileDescription, AppHInstance,

	AppLegalCopyright, AppLegalTrademarks, AppMajor, AppMinor, AppRevision, AppTitle, AppEXENAME, AppPath, ScriptEngineMajorVersion, ScriptEngineMinorVersion, ScriptEngineBuildVersion
--	--

INDEX

' , 176
" ", 42
&, 16
ABS, 56
AddItem, 143
ADDOBJECT, 19, 57, 102,
143, 159, 163
Alignment, 168
AND, 19, 60
AppHInstance, 239
ARRAY, 61
ASC, 62
ASCB, 62
ASCW, 62
ATN, 63
BackColor, 168
BorderStyle, 168
Bottom, 168
BREAK, 37, 64
BYE, 65
CALL, 66
Caption, 168
CaptionColor, 121
CBOOL, 76
CBYTE, 76
CCUR, 76
CDATE, 76
CDBL, 76
CHAIN, 67
Change, 100
CheckBox, 69
CHR, 71
CHRB, 71
CHRW, 71
CINT, 76
CLASS, 72
Clear, 143
Click, 100
CLNG, 76
close button, 27
Cls, 164
Coding Conventions, 233
ComboBox, 30, 73
command line, 54
CommandButton, 74
comment, 14
compile, 54
CONST, 75
Copy, 32
COS, 78
CREATESHORTCUT, 79
Creating a Program, 31
CSNG, 76
CSTR, 76
CurrentPath, 80
Cut, 32
Data Types, 14
Array, 16
Boolean, 15
Color, 15
Numeric, 15
String, 16
DATE, 81, 82, 168
DATEADD, 83
DATEDIFF, 84
DATEPART, 86
DATESERIAL, 87
DATEVALUE, 88
DAY, 89
DbClick, 100
Debugging a Program, 34
DECLARE, 90
DIM, 91
Dir To Include, 169
DO...LOOP, 93
DOEVENTS, 92
DrawCircle, 164
DrawLine, 164
DrawPicture, 164
DrawPoint, 164
DrawText, 164
DrawWidth, 163
DropDown, 100
Editing a Program, 32
Elements, 14, 16
Enabled, 169
Encrypted, 169
EQV, 19, 95

ERASE, 96	ABS, 56
Err, 20, 97	ARRAY, 61
error handler, 235	ASC, 62
errors, 34	ASCB, 62
compile_time, 35	ASCW, 62
logic, 35	ATN, 63
run-time, 35	CBOOL, 76
ESCAPE, 98	CBYTE, 76
EVAL, 99	CCUR, 76
Events, 20	CDATE, 76
Change, 100	CDBL, 76
Click, 100	CHR, 71
DbClick, 100	CHRB, 71
DropDown, 100	CHRW, 71
GotFocus, 100	CINT, 76
EXECUTE, 13, 37, 102	CLNG, 76
EXECUTEGLOBAL, 103	COS, 78
EXIT, 104	CREATESHORTCUT,
EXP, 105	79
ExpandedHeight, 169	CSNG, 76
Expressions, 16	CSTR, 76
FillColor, 163	DATE, 81
FillStyle, 163	DATEADD, 83
FILTER, 106	DATEDIFF, 84
Find, 32	DATEPART, 86
Find All, 32	DATESERIAL, 87
Find Next, 32	DATEVALUE, 88
firstdayofweek, 84	DAY, 89
firstweekofyear, 84	ESCAPE, 98
FIX, 108	EVAL, 99
FontBold, 169	EXP, 105
FontItalic, 169	FILTER, 106
FontName, 169	FIX, 108
FontSize, 169	FORMATCURRENCY
FontStrikethru, 169	, 111
FontTransparent, 163	FORMATDATETIME,
FontUnderline, 169	111
FontWeight, 169	FORMATNUMBER,
FOR EACH...NEXT, 110	111
FOR...NEXT, 109	FORMATPERCENT,
ForeColor, 169	111
Form_Load, 21, 23, 43	GETCOMMANDLINE,
Form_Unload, 21, 43	116
FORMATCURRENCY, 111	GETLOCALE, 117
FORMATDATETIME, 111	GETREF, 118
FORMATNUMBER, 111	GETRESOURCE, 119
FORMATPERCENT, 111	GETSERIALNUMBER
Forms, 19	, 120
Frame, 113	HEX, 122
FUNCTION, 19, 36, 114	HOUR, 123
Functions	IMP, 125

INPUTBOX, 126	TRIM, 217
INSTR, 127	TYPENAME, 218
INSTRREV, 127	UBOUND, 220
INT, 128	UCASE, 221
ISARRAY, 130	UNESCAPE, 98
ISDATE, 130	VARTYPE, 223
ISEMPTY, 130	WAITCURSOR, 225
ISNULL, 130	WEEKDAY, 227
ISNUMERIC, 130	WEEKDAYNAME,
ISOBJECT, 130	228
JOIN, 131	YEAR, 232
LBOUND, 136	GETCOMMANDLINE, 116
LCASE, 137	GETLOCALE, 117
LEFT, 138	GETREF, 118
LEFTB, 138	GETRESOURCE, 119
LEN, 139	GETSERIALNUMBER, 120
LENB, 139	Global Properties
LOG, 141	CurrentPath, 80
LTRIM, 142	KeyboardStatus, 132
MID, 145	KeyboardStatusChang
MIDB, 145	ed, 132
MINUTE, 146	KeyPreview, 133
MONTH, 148	NSBVersion, 155
MONTHNAME, 149	GotFocus, 100
MSGBOX, 150	GRADIENTBUTTON, 121
OCT, 156	GradientColor1, 121
PLAYSOUND, 166	GradientColor2, 121
REPLACE, 177	GradientStyle, 121
RGB, 178	Group, 159, 169
RIGHT, 179	Height, 169
RIGHTB, 179	help button, 27
RND, 180	HEX, 122
ROUND, 181	Hi Res Aware, 22, 170
RTRIM, 182	Hide, 143
SECOND, 186	HOUR, 123
SETLOCALE, 117	HScrollbar, 185
SGN, 195	HTMLView, 58
SIN, 199	Hwnd, 170
SPACE, 201	Icon, 170
SPECIALFOLDER,	Icons, 23
202	IF...THEN...ELSE, 124
SPLIT, 203	IMP, 19, 125
SQR, 204	INPUTBOX, 126
STRCOMP, 205	Installation, 11
STRING, 206	INSTR, 127
STRREVERSE, 207	INSTRREV, 127
SYSINFO, 210	INT, 128
TAN, 211	IntegralHeight, 170
TIME, 213	IS, 129
TIMESERIAL, 215	ISARRAY, 130
TIMEVALUE, 216	ISDATE, 130

ISEMPY, 130
 ISNULL, 130
 ISNUMERIC, 130
 ISOBJECT, 130
 JOIN, 131
 KeyboardStatus, 132
 KeyboardStatusChanged, 132
 KeyDown, 100, 164
 KeyPress, 100, 164
 KeyPreview, 133
 KeyUp, 100, 164
 KEYWORD, 14
 KILLFOCUS, 134
 Label, 135
 LBOUND, 136
 LCASE, 137
 LEFT, 138
 LEFTB, 138
 LEN, 139
 LENB, 139
 License, 57
 List, 170
 ListBox, 30, 140
 ListCount, 170
 ListIndex, 170
 Literals, 14
 LOG, 141
 LongFormat, 170
 LostFocus, 100
 LTRIM, 142
 MaxLength, 171
 Menu Editor, 30
 Menus, 25
 Methods, 20
 AddItem, 143
 Clear, 143
 Hide, 143
 Move, 143
 RemoveItem, 143
 SetFocus, 143
 Show, 143
 MID, 145
 MIDB, 145
 MINUTE, 146
 MOD, 17, 147
 Modules, 19, 22
 MONTH, 148
 MONTHNAME, 149
 MouseDown, 164
 MouseMove, 165
 MouseUp, 165
 MSGBOX, 35, 150
 MultiLine, 171
 Multi-Line Statements, 14
 Name, 171
 Naming guidelines, 233
 NewIndex, 171
 NOT, 19, 153
 notation conventions, 13
 NOW, 154
 NSBVersion, 155
 NSEXECUTE, 102
 Objects
 ADDOBJECT, 159, 163
 CheckBox, 69
 ComboBox, 73
 CommandButton, 74
 Date, 82
 Err, 97
 Frame, 113
 GRADIENTBUTTON, 121
 HScrollbar, 185
 Label, 135
 ListBox, 140
 OUTPUT, 162
 TextBox, 212
 Time, 214
 TriButton, 69
 VScrollBar, 185
 OCT, 156
 ON ERROR, 157, 235
 Operator
 XOR, 231
 Operators, 16
 AND, 60
 Arithmetic, 17
 Boolean, 18
 EQV, 95
 IS, 129
 MOD, 147
 NOT, 153
 OR, 161
 Relational, 18
 OPTION EXPLICIT, 37, 158, 235
 OR, 19, 161
 OUTPUT, 162, 167
 Output object, 20, 22, 133
 Output Window, 27

Output_close, 162	SelText, 172
Output_Close event, 22	Sorted, 172
Output_Size, 22, 162	SpinBox, 172
Overview, 33	Style, 172
Parent, 57	Tabstop, 172
parenthesis, 19	Tag, 172
ParentHWnd, 171	Text, 172
Paste, 32	Timer, 172
Picture, 163	Top, 172
PictureBox, 20, 22	Value, 173
PLAYSOUND, 166	Visible, 173
PRINT, 13, 35, 167	Width, 173
PRIVATE, 75	WindowLong, 173
Program Editor, 25, 32	Property Editor, 30
Projects, 19	PUBLIC, 75
Properties, 19	RadioButtons, 159
Alignment, 168	RANDOMIZE, 174
BackColor, 168	REDIM, 175
BorderStyle, 168	Refresh, 164
Bottom, 168	REM, 176
Caption, 168	Removeltem, 143
Date, 168	REPLACE, 177
Enabled, 169	RGB, 178
Encrypted, 169	Right, 171, 179
ExpandedHeight, 169	RIGHTB, 179
FontBold, 169	RND, 180
FontItalic, 169	ROUND, 181
FontName, 169	RTRIM, 182
FontSize, 169	RUNAPPATEVENT, 183
FontStrikethru, 169	RUNAPPATTIME, 184
FontUnderline, 169	Running a Program, 34
FontWeight, 169	run-time, 234
ForeColor, 169	Sample programs, 9
Group, 169	Saving and Loading a
Height, 169	Program, 38
Hwnd, 170	ScaleHeight, 163
Icon, 170	ScaleLeft, 163
IntegralHeight, 170	ScaleMode, 163
Left, 170	ScaleTop, 163
List, 170	ScaleWidth, 163
ListCount, 170	ScaleX, 164
ListIndex, 170	ScaleY, 164
LongFormat, 170	scrollbar, 185
MaxLength, 171	Scrollbars, 171
MultiLine, 171	SECOND, 186
NewIndex, 171	seed, 180
ParentHWnd, 171	SELECT CASE, 187
Right, 171	SelLength, 172
Scrollbars, 171	SelStart, 172
SelLength, 172	SelText, 172
SelStart, 172	SENDKEY, 189

SENDMESSAGE, 190
 SENDMESSAGESTRING, 190
 Serial Number, 11
 SET, 18, 191
 SetFocus, 143
 SETLOCALE, 117
 SETMENU, 27, 192
 SETPARENT, 194
 SetScale, 164
 SGN, 195
 SHELLEXECUTE, 196
 Show, 143
 SHOWFULLSCREEN, 197
 ShowOKButton, 22, 198
 SIN, 199
 SLEEP, 200
 Sorted, 172
 SPACE, 201
 SPECIALFOLDER, 202
 SpinBox, 172
 SPLIT, 203
 SQR, 204
 Statement, 14
 Statement continuation, **42**
 Statements
 ADDOBJECT, 57
 BREAK, 64
 BYE, 65
 CALL, 66
 CHAIN, 67
 CLASS, 72
 CONST, 75
 DECLARE, 90
 DIM, 91
 DO...LOOP, 93
 DOEVENTS, 92
 ERASE, 96
 EXECUTE, 102
 EXECUTEGLOBAL, 103
 EXIT, 104
 FOR EACH...NEXT, 110
 FOR...NEXT, 109
 FUNCTION, 114
 IF...THEN...ELSE, 124
 KILLFOCUS, 134
 ON ERROR, 157
 OPTION EXPLICIT, 158
 PRINT, 167
 RANDOMIZE, 174
 REDIM, 175
 REM, 176
 RUNAPPATEVENT, 183
 RUNAPPATTIME, 184
 SELECT CASE, 187
 SENDKEY, 189
 SENDMESSAGE, 190
 SENDMESSAGESTRING, 190
 SET, 191
 SETMENU, 192
 SETPARENT, 194
 SHELLEXECUTE, 196
 SHOWFULLSCREEN, 197
 SHOWOKBUTTON, 198
 SLEEP, 200
 SUB, 208
 UPDATESCREEN, 222
 WAVEVOLUME, 226
 WHILE...WEND, 230
 WITH, 229
 Step, 37
 STRCOMP, 205
 STRING, 206
 STRREVERSE, 207
 Style, 172
 SUB, 36, 208
 SUB procedures, 20
 SYSINFO, 210
 System Requirements, 10
 Tabstop, 172
 Tag, 163, 172
 TAN, 211
 Text, 172
 text coloring, **42**
 TextBox, 212
 TextHeight, 164
 TextWidth, 164
 TIME, 213, 214
 Timer, 100, 172
 TIMESERIAL, 215
 TIMEVALUE, 216
 Top, 172
 Trace, 37

TriButton, 69	Visible, 173
TRIM, 217	VScrollBar, 185
TRUE, 15	WAITCURSOR, 225
TYPENAME, 218	WAVEVOLUME, 226
UBOUND, 220	WEEKDAY, 227
UCASE, 221	WEEKDAYNAME, 228
UNESCAPE, 98	WHILE...WEND, 230
UPDATESCREEN, 222	Width, 173
Value, 173	WindowLong, 173
Variables, 14	WITH, 229
Names, 16	XOR, 19, 231
VARTYPE, 223, 235	YEAR, 232

USER'S COMMENT FORM

Please use this form only to identify publication errors or to request changes in publications. Please let us know if you would like a reply. Return to:

NS BASIC Corporation
71 Hill Crescent
Toronto, Canada M1M 1J3
fax (416) 264-5888

An alternative method of contacting NS Basic about publication errors and requested changes is to send email to:

support@nsbasic.com

The message subject should be the title of the publication followed by the printing date (from the bottom of the first page)

Page	Comments